Hes·so
University of Applied Sciences
Western Switzerland

Travail de Diplôme

Pour obtenir le grade de

Bachelor

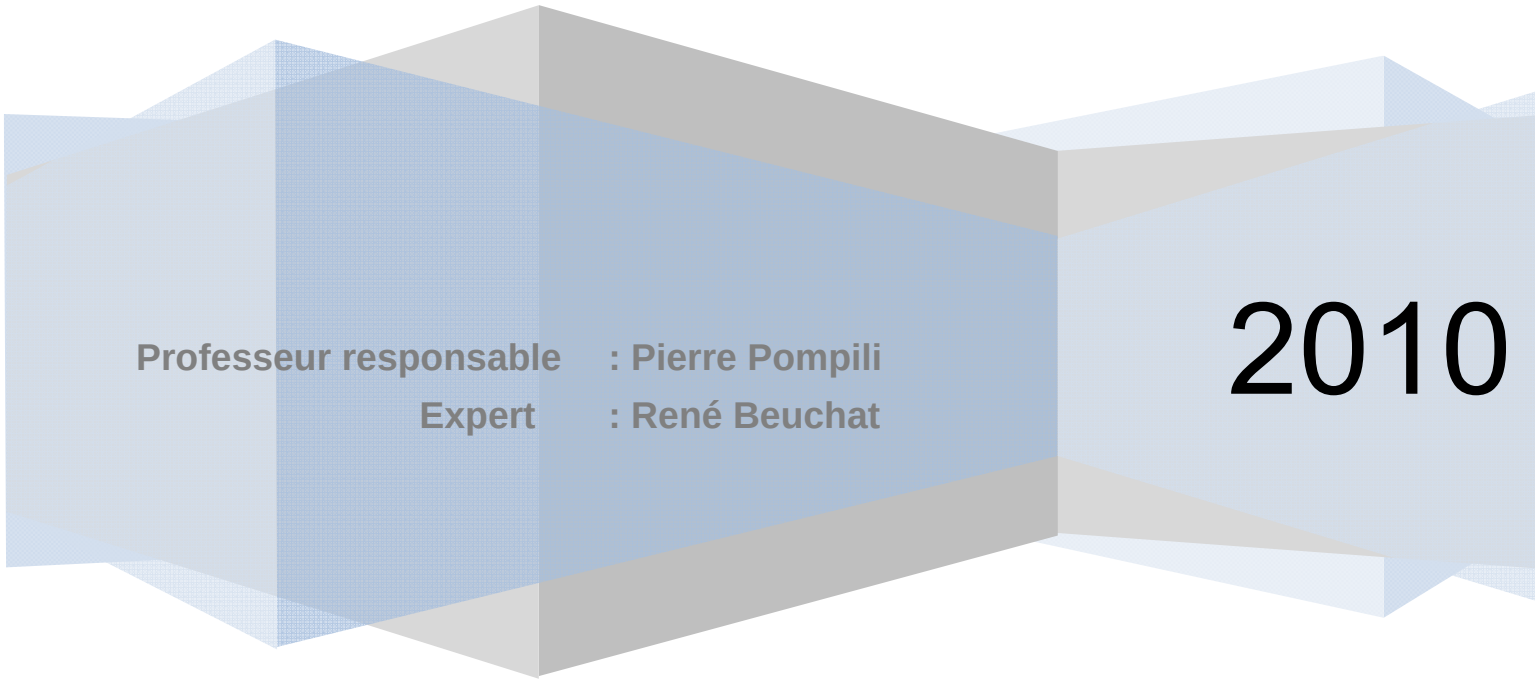
Filière : Systèmes Industriels

Orientation : Infotronique

Réalisé au sein de la section Infotronique de l'HES-SO Sion

Bi-Cortex-M3

Castellaro Christian



Professeur responsable : Pierre Pompili

Expert : René Beuchat

2010

HES-SO Valais

SI	TV
X	X

Données du travail de diplôme
Daten der Diplomarbeit

FO 1.2.02.07.AB
pof/31/01/2009

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2009/10**	No TD / *Nr. DA* **it/2010/20**
Mandant / *Auftraggeber* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire	Etudiant / *Student* **Christian Castellaro**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire
Professeur / *Dozent* **Pierre Pompili**	Expert / *Experte* *(données complètes)* **René Beuchat**	
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein		

Titre / *Titel*

Bi-Cortex-M3

Description et Objectifs / *Beschreibung und Ziele*

Les processeurs de dernière génération intègrent dans leur cœur une technologie RISC, des unités d'optimisation des sous-systèmes. Le cœur ARM Cortex-M3 est en passe de devenir le processeur le plus utilisé dans les systèmes embarqués 32bits.

Ce projet consiste à mettre en œuvre un démonstrateur Cortex-M3, puis de développer une carte-mère avec deux processeurs de type Cortex-M3, avec comme but ultime le portage d'un noyau temps réel RTEMS multiprocesseurs.

- Mettre en place les outils logiciels pour le développement d'applications pour un ARM Cortex-M3
- Étudier les mécanismes de gestion de la mémoire offerts dans un ARM Cortex-M3
- Définir l'architecture la plus appropriée pour un système embarqué à deux cœurs ARM Cortex-M3
- Concevoir l'électronique d'un système embarqué à deux cœurs ARM Cortex-M3
- Développer une application mettant en valeur le potentiel des deux cœurs ARM Cortex-M3
- Porter un noyau temps réel RTEMS multiprocesseurs sur la cible, uniquement si le temps le permet.

Délais / *Termine*

Attribution du thème / *Ausgabe des Auftrags:* 22.02.2010	Remise du rapport / *Abgabe des Schlussberichts:* 12.07.2010, 12:00
Remise du rapport intermédiaire / *Zwischenbericht:* 07.05.2010, 17:00	Exposition publique / *Ausstellung Diplomarbeiten:* 27.08.2010
Défense intermédiaire / *Zwischenverteidigung:* 21.05.2010	Défense orale / *Mündliche Verteidigung:* Semaine 35 / *Woche 35*

Signature ou visa / Unterschrift oder Visum

Responsable de la filière
Leiter des Studiengangs: .....

[1] Etudiant/*Student*: .....

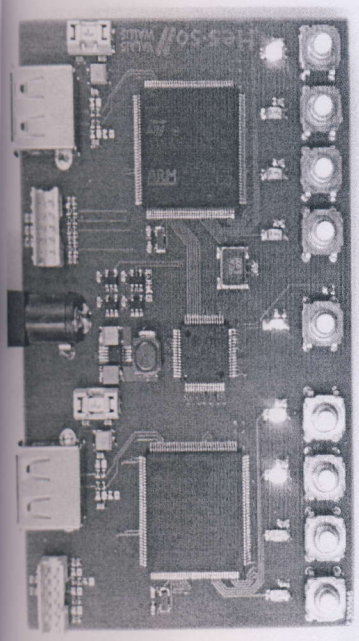
[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement le caractère confidentiel du travail de diplôme qui lui est confié et des informations mises à sa disposition.*
Durch seine Unterschrift verpflichtet sich der Student, die Vertraulichkeit der Diplomarbeit und der dafür zur Verfügung gestellten Informationen zu wahren.

Rapport reçu le / *Schlussbericht erhalten am* ..... 12 JUIL. 2010 Visa du secrétariat / *Visum des Sekretariats*

Hes·so VALAIS WALLIS
Sciences de l'ingénieur·e

Hes·so // VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Bi-Cortex-M3



Diplômant/e Christian Castellaro

Travail de diplôme
| édition 2010 |

Filière
Systèmes industriels

Domaine d'application
Infotronics

Professeur responsable
Pierre Pompili
Pierre.Pompili@hevs.ch

Objectif du projet

Ce projet consiste à développer une carte mère avec deux processeurs de type Cortex-M3, concevoir une application mettant en valeur le potentiel des deux cœurs ARM Cortex-M3 et porter un noyau temps réel RTEMS multiprocesseurs.

Méthodes | Expériences | Résultats

En premier lieu, les différentes architectures mémoires possibles pour concevoir un système multiprocesseurs ont été évaluées afin de déterminer la plus adaptée à ce projet. La carte mère prototypée aura une architecture à mémoire partagée, avec dans chaque processeur une mémoire privée.

L'environnement de développement Eclipse a ensuite être configuré avec une toolchain capable de programmer et débugger des applications sur chacun des cœurs.

Una application de calcul des décimales de π basée sur l'algorithme de Bailey – Borwein – Plouffe optimisée pour utiliser deux cœurs a été programmée.
Le gain en performances obtenu avec cette application est proche de la valeur optimale de 2 quand le second cœur est utilisé.

Le portage de l'OS temps réel RTEMS apporte une gestion native des systèmes multiprocesseurs garantissant une utilisation maximale des ressources matérielles. Seule une partie du portage a été réalisée.

Tous les éléments nécessaires à finaliser le portage sont en place.

HES-SO Valais
Route du Rawyl 47
1950 Sion

Tél. 027 606 85 11
URL www.hevs.ch

Index

Glossaire

Les mots apparaissant en gras dans ce document (uniquement la première fois qu'ils apparaissent) sont définis dans ce glossaire.

finesse de gravure : définit la grandeur de la plus petite taille gérable dans le processus de fabrication d'une puce. Actuellement, la taille la plus petite maitrisée est de 32nm, cela représente quelques centaines d'atomes!

temps réel : se dit d'une application qui garantit que la réponse à un évènement intervient TOUJOURS avant une limite définie. Par exemple pour le temps réel dur cette limite doit être inférieure à 1 milliseconde.

environnement de développement (IDE) : programme capable de réunir dans une même interface graphique des programmes qui sont normalement indépendants. Ces environnements sont souvent compliqués à configurer mais une fois cette étape passée le gain en temps et en confort d'utilisation est considérable.

toolchain : se dit d'une suite de programmes conçus pour être utilisés ensemble. Le fait de morceler en plusieurs programmes une fonctionnalité permet une grande modularité, une portabilité simplifiée et une plus grande compatibilité avec différentes "cibles" au sens large du terme.

cross-developement : nom donné à la manière de plus en plus courante de développer des applications sur un système hôte qui n'est pas du tout celui sur lequel tourneront les programmes, appelé système cible.

remote debugging : technique qui permet de contrôler par un système hôte l'exécution d'une application sur une cible. Les fonctionnalités nécessaires au debuggage sont :

- les breakpoints, qui stoppent l'exécution du code à une instruction précise
- l'accès aux données contenues dans les registres du cœur
- l'accès aux données dans toutes les mémoires du système cible (ram et flash)

gestion des priorités : pour pouvoir garantir une réponse temps réel à un événement, le système doit pouvoir gérer une priorisation des tâches, de sorte à stopper l'exécution d'une tache secondaire pour pouvoir en effectuer une prioritaire.

thread : nom donné à une application ou une partie d'application lorsque plusieurs sont en exécution dans un même système. L'attribution planifiée des ressources à ces threads ainsi que leur priorisation sont les concepts fondamentaux de la conception temps réel.

System-on-Chip : extension de la notion de processeur. Un SoC est une puce embarquant quasiment un système informatique complet (cœur, mémoire, périphériques, coprocesseurs, dsp, etc.)

dongle : interface matérielle utilisée en cross-developement permettant de transformer les signaux envoyés à la cible par l'hôte. L'adaptation des niveaux électriques ainsi que la gestion du protocole utilisé par la cible est effectué dans ces dongles.

Low ESR (Equivalent Series Resistance) : se dit d'un condensateur dont la résistance série équivalente est faible. Sert à atténuer l'ondulation sur la tension qu'il découple.

Ferrite : composant électrique de la famille des inductances spécialement conçu pour atténuer les hautes fréquences et laisser passer les basses. Contribue à la stabilité d'une alimentation continue.

sémaphore : outil logiciel servant à garantir l'accès unique (ou contrôlé) à une ressource. Par exemple plusieurs applications peuvent lire une donnée au même emplacement mémoire sans besoin de contrôle. Par contre une seule peut y écrire à un moment donné. Le sémaphore permet cette gestion.

stack : bloc de mémoire réservé au déroulement d'une application.

Mips : acronyme pour Million d'Instructions Par Seconde. C'est une mesure permettant d'avoir une idée de la puissance de calcul d'un processeur. Ce n'est pas une mesure absolue, le concept d'instructions d'un processeur à l'autre pouvant considérablement changer. Cette mesure est efficace dans une même famille de processeurs.

bus AHB : bus principal du cœur Cortex-M3 en général cadencé à la même fréquence que le cœur.

bus APB1: bus de périphériques rapide du cœur Cortex-M3 en général cadencé à la même fréquence que le cœur.

bus APB2: bus de périphériques lent du cœur Cortex-M3 en général cadencé à la moitié de la fréquence du cœur.

poller : action d'aller contrôler qqch à intervalle de temps réguliers et définis. Les valeurs des flags sont typiquement des éléments qui sont pollés dans une machine d'états-transition.

processeur : puce qui contient un cœur, souvent des mémoires (caches, flash ou ram et qques périphériques).

cœur : élément principal du processeur où tous les calculs sont faits. Il contient en général une **ALU**, un bus de données, un bus d'adresses et des registres fondamentaux comme le PC (program counter).

waitstates : intervalles de temps ajoutés par un processeur au cours d'une instruction pour attendre qu'une ressource soit libérée

burst mode : méthode d'accès en mémoire qui permet en ne changeant pas la valeur sur le bus d'adresses d'effectuer une série de lectures /écritures consécutives.

mémoire synchrone : mémoire disposant d'une entrée d'horloge qui nécessite d'être pilotée de façon synchrone par un contrôleur.

interruption : événement particulier qui interromps le déroulement actuel de la tâche qui s'exécute sur un processeur pour être traité en priorité.

caches : mémoires souvent comprises dans les processeurs. De petite taille mais extrêmement rapides elles servent à limiter les accès à la ram qui est plus lente.

chipenable : signal émit par un contrôleur pour indiquer à un périphériques que les action suivantes lui sont destinées

serveur daemon : programme tournant en tâche de fond et attendant que des requêtes lui soient adressées

système d'exploitation : Logiciel souvent très complexe permettant entre autre la gestion du multitâches, des priorités et abstrayant la couche matérielle en lui donnant accès par une interface logicielle appelée driver.

ALU : pour Arithmetical and Logical Unit, partie du cœur d'un processeur qui effectue les calculs et les opérations logiques.

Table des Figures

1 Introduction

1.1 Plus de performances

L'évolution et l'amélioration des logiciels ainsi que les nouveaux besoins des utilisateurs amènent les concepteurs de systèmes informatiques à fournir toujours plus de performances. Réactivité accrue, temps de chargements minimaux et puissance de calcul conséquente sont autant d'exigences que les nouveaux systèmes se doivent de satisfaire.

Avec le niveau de complexité et d'optimisation du code atteint par les compilateurs actuellement, les solutions logicielles pour améliorer les performances des applications sont inefficaces si le logiciel a été bien conçu dans un premier temps. Augmenter la puissance de traitement du hardware exécutant le code est donc devenu la seule manière d'atteindre ces objectifs.

Pendant de nombreuses années, l'accroissement des performances a été obtenu essentiellement par la montée en fréquence des **processeurs**. Cependant depuis quelques années, une nouvelle solution prometteuse a été mise en œuvre, la multiplication des processeurs au sein d'un même système.

1.2 Multiplication des cœurs

Accroitre la vitesse de fonctionnement des puces est une solution efficace car l'amélioration des performances est quasi proportionnelle à l'augmentation de la fréquence. De plus, aucune modification du logiciel n'est nécessaire afin d'exploiter ce gain. Cependant, cette solution miracle a atteint ses limites. La volonté de multiplier le nombre d'unités de traitement s'explique clairement au travers de la Figure 1.

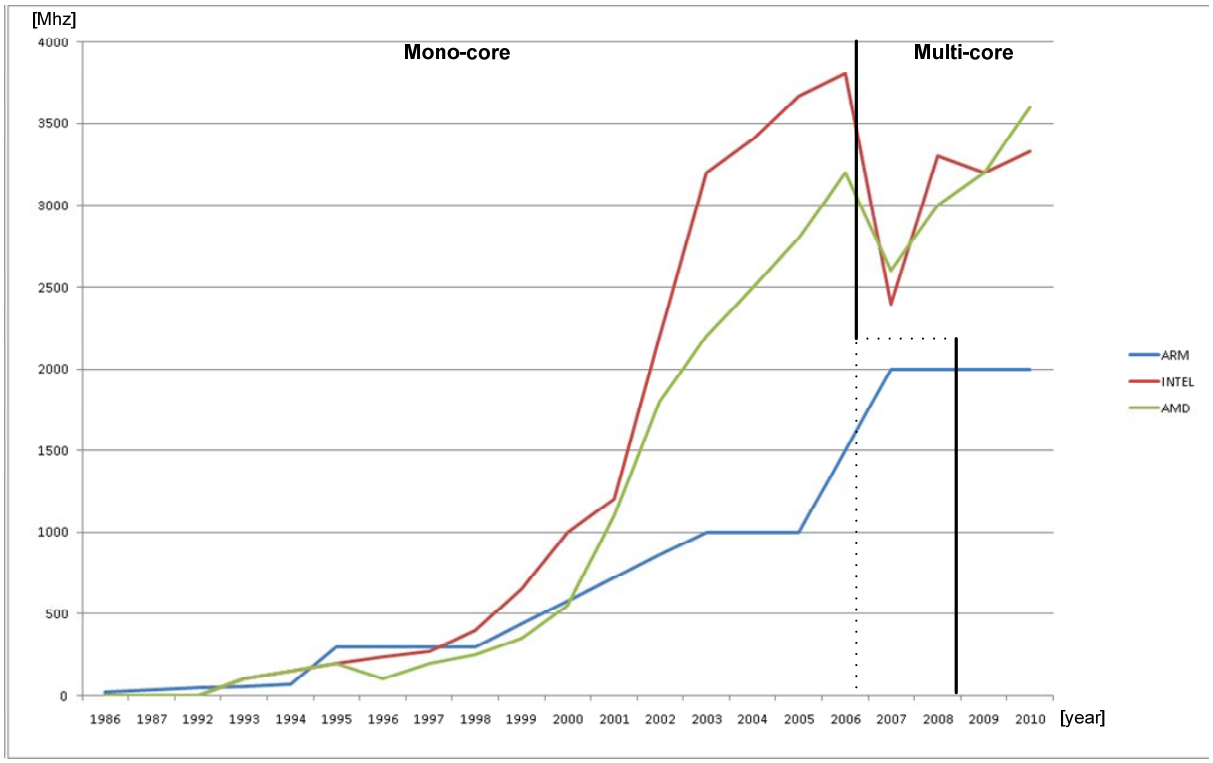


Figure 1 : Evolution des fréquences en fonction de l'année

De 2000 à 2003, une évolution de 300% des fréquences a été obtenue chez Intel alors que de 2003 à 2006 seule une évolution de 120% a pu être accomplie. Les raisons de cette stagnation sont physiques. Sans entrer trop dans les détails, la diminution de la **finesse de gravure** (nécessaire à accroitre la fréquence d'horloge) entraine deux effets nuisibles qui se cumulent :

- Un accroissement des courants de fuite à l'intérieur de la puce dus à une couche isolante insuffisante entre les transistors.
- Une diminution de la taille de la puce et donc de la surface disponible pour dissiper la chaleur produite.

Ces deux effets cumulés entrainent une augmentation exponentielle de la chaleur produite par la puce et de sa consommation électrique, rendant les moyens conventionnels de refroidissement inutilisables. Cela explique l'avènement des systèmes multi processeurs.

Il parait évident que si deux processeurs travaillent sur la même tâche les performances sont doublées. La réalité n'est bien sur pas si simple. Bien que la puissance théorique matérielle du système soit effectivement doublée, le logiciel doit être conçu spécifiquement pour en tirer parti. C'est pour cette raison que l'industrie a attendu le plus longtemps possible avant de se lancer dans cette voie : la très grande majorité des applications ne sont pas prévue pour utiliser plusieurs unités de traitement en parallèle.

La conception de logiciels doit donc passer d'une logique séquentielle des opérations à une logique paralléllisable et cela n'est pas toujours facile voir même possible. C'est cependant devenu une nécessité.

1.3 Systèmes embarqués

Bien que les fréquences des puces utilisées dans les systèmes embarqués soient encore loin des limites observées plus haut, les contraintes de consommation électriques et aussi dans une moindre mesure de dissipation thermique rendent les solutions multi processeurs intéressantes. Cependant, bien que la Figure 1 montre que la démocratisation des puces multi processeurs s'est faite entre 2006 et 2007 pour les ordinateurs conventionnels, cette évolution ne commence qu'actuellement pour les systèmes embarqués.

La raison principale est le besoin récent et croissant des utilisateurs d'avoir constamment sur eux une machine quasiment capable de remplacer un ordinateur traditionnel. C'est pour répondre efficacement à cette demande que les systèmes embarqués multi processeurs sont apparus. C'est donc dans ce contexte qu'est né ce travail de diplôme, nommé Bi-Cortex M3.

2 Objectifs

L'objectif principal de ce travail de diplôme est d'acquérir de l'expérience dans le développement hardware et software de systèmes embarqués **temps réels** multi processeurs. L'utilisation de processeurs à base de l'architecture Cortex-M3 de l'entreprise ARM a été motivée par sa conception spécifiquement étudiée pour le temps réel.

Il faudra en premier lieu concevoir une carte de démonstration mettant en œuvre deux processeurs. Pour ce faire, une spécification matérielle complète de la carte devra être proposée. Le choix de l'architecture mémoire du système servira de base autour de laquelle la carte sera conçue. Il faudra ensuite choisir les différents composants qui seront le plus à même de d'implémenter la spécification et réaliser la schématique électrique du système pour fabriquer un prototype fonctionnel.

L'étape suivante consistera en la mise en place d'un **environnement de développement** permettant de concevoir et de debugger des applications spécifiques en langage C/C++ pour la carte de démonstration. Les problématiques de la **toolchain** nécessaire au **cross-developement** et au **remote debugging** seront abordées.

Une fois l'environnement de développement mis en place, une application utilisant au mieux les ressources du système sera conçue. Il sera ainsi possible de comparer les performances du système lorsqu'un seul ou les deux processeurs sont utilisés.

Enfin, il restera à effectuer le portage d'un **système d'exploitation** (OS) temps réel sur le système afin de simplifier le développement ultérieur d'applications, d'optimiser l'utilisation des ressources du système et de pouvoir respecter des contraintes d'exécution temps réel à l'aide, entre autre, des outils de gestion des **threads** et de **gestion des priorités** offertes par un OS.

3 Développement hardware

Ce chapitre traite de tous les aspects matériels de ce projet, de la spécification de la carte au choix des composants en passant par les détails de la schématique électrique. Il se termine par un descriptif des tests effectués pour vérifier le bon fonctionnement de la carte ainsi que de leurs résultats. Tous les composants utilisés pour produire la carte se trouvent dans la liste des composants fournie en annexe I. Le cd-rom contient les datasheets de tous les composants particuliers dans le répertoire /hardware/datasheets_composants. Un rapide coup d'œil à la Figure 18 permet de situer chaque partie de la carte, l'annexe de la schématique électrique correspondante et le chapitre dans lequel est traitée cette partie.

3.1 Spécification de la carte

Les exigences que doit remplir la carte de démonstration ont été décidées en collaboration avec le responsable de projet, M. Pierre Pompili, afin de garantir un minimum de fonctionnalités. La carte doit disposer de :

- 2 processeurs à **cœur** Cortex-M3 avec un canal de communication à déterminer
- 1 port **JTAG** par processeur avec possibilité matérielle de sérialiser la chaine par la suite
- 4 boutons et 4 leds d'interaction utilisateur par processeur
- 1 led témoin pour l'état d'alimentation
- 1 port USB par processeur
- 1 bouton de Reset commun pour les deux processeurs
- énergie fournie par une alimentation externe à 5V DC

Voici un schéma bloc de référence illustrant cette spécification (Figure 2).

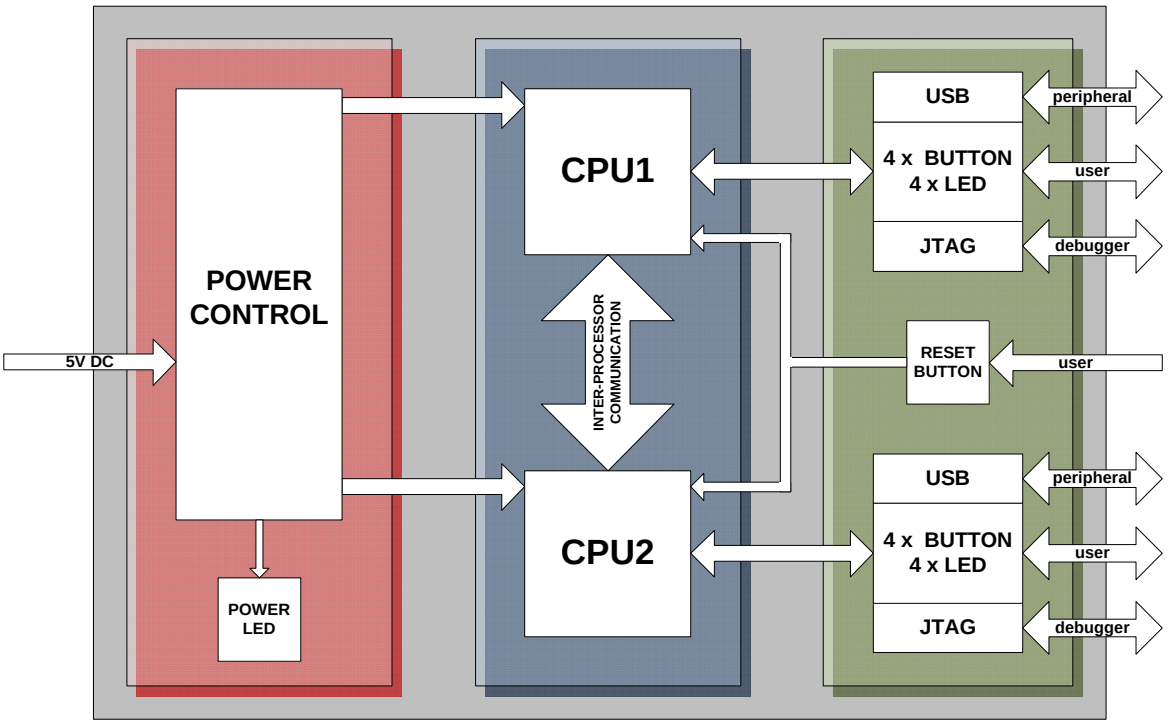


Figure 2 : Schéma bloc général de la carte

Un schéma bloc plus détaillé sera proposé quand les processeurs et le canal de communication entre eux seront définis (voir Figure 7 chapitre 3.2.2).

3.2 Conception de la carte

Cette partie décrit les différents choix qui ont du être fait afin de répondre au mieux à la spécification de la carte. La première étape est la définition de l'architecture mémoire du système, qui va être le squelette sur lequel toutes les autres parties vont reposer. Une autre étape importante est le choix du processeur utilisé, qui va conditionner une bonne partie de la suite du développement. Toutes les fonctionnalités de la carte sont abordées. Certaines, triviales comme les entrées/sorties, le sont de façon rapide alors que d'autres (JTAG ou connexion de la mémoire) sont très détaillées.

3.2.1 Architecture mémoire du système

Tout système informatique comporte de la mémoire. En effet autant complexe que soit l'application exécutée par un processeur, elle peut être résumée à lire des données dans une mémoire, les traiter et les réécrire dans une mémoire. Cet élément est donc fondamental dans la conception d'un système. Sa vitesse, sa taille et aussi sa localisation sont autant de paramètres qu'il est important d'étudier et ce d'autant plus dans un système multi processeurs.

Pour exploiter efficacement les ressources disponibles, les threads qui s'exécutent sur les différents cœurs doivent pouvoir communiquer entre eux, s'échanger des résultats et accéder à des données communes. L'architecture mémoire doit donc permettre tous ces échanges. Il existe deux familles d'architectures pour concevoir un système multi processeurs, l'architecture à mémoire distribuée et l'architecture à mémoire partagée.

3.2.1.1 MÉMOIRE DISTRIBUÉE

La Figure 3 montre un schéma d'architecture distribuée. Chaque unité de traitement possède une mémoire privée qu'elle est la seule à accéder et elles communiquent entre elles via un réseau d'interconnections par échange de messages. Les avantages de cette répartition de la mémoire sont d'une part son extensibilité à plusieurs dizaines de cœurs, sa répartition des échanges entre processeurs par des canaux différents et la rapidité de l'accès à la mémoire privée. Son plus gros désavantage est par contre une gestion très compliquée et lourde de la cohérence des données dans tout le système. Par exemple si une variable est utilisées par plusieurs processeurs et que l'un d'eux en modifie la valeur dans sa mémoire privée il faut la mettre à jour dans toutes les autres mémoires où elle est utilisée. Ce mécanisme est souvent couteux en temps et nécessite parfois l'ajout de hardware pour être effectué. Il est à noter que le protocole gérant le réseau d'interconnections est souvent complexe à concevoir.

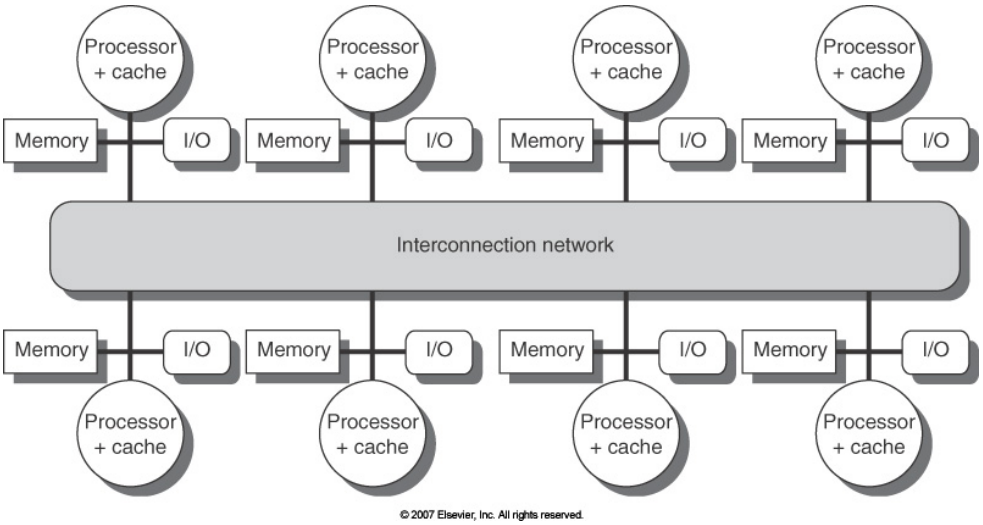


Figure 3 : Architecture à mémoire distribuée

3.2.1.2 MÉMOIRE PARTAGÉE

Un exemple d'architecture à mémoire partagée est visible en Figure 4. Ici une mémoire globale commune à tous les processeurs est accédée au travers d'un bus commun. La mémoire commune devient donc le vecteur de communication entre les processeurs. Le point fort de cette architecture est sa simplicité et sa rapidité de mise en œuvre. En effet, il n'y a plus d'intercommunication directe entre les processeurs donc plus de protocole complexe à implémenter et le problème de la cohérence des données bien que toujours présent (à cause de la présence de **caches** dans la plupart des processeurs) peut-être résolu de manière plus aisée. Cette simplicité a cependant un coût. Le bus commun d'accès à la mémoire devient très vite un goulet d'étranglement si le nombre d'unités de traitement dépasse la dizaine et il faut tout de même implémenter un protocole d'accès au bus qui garantisse qu'un seul cœur l'utilise à un instant donné, ralentissant l'accès à la mémoire.

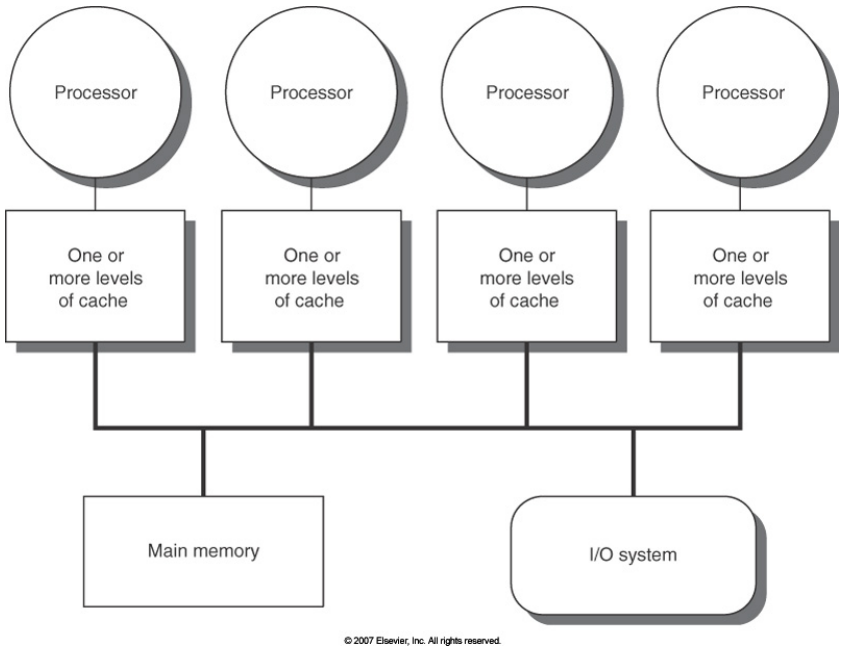


Figure 4 : Architecture à mémoire partagée

3.2.1.3 OPTION RETENUE : MÉMOIRE PARTAGÉE DUAL-PORT

La carte à développer dans le cadre de ce travail de diplôme ne devant comporter que deux cœurs, le choix d'une architecture à mémoire partagée semble être le choix le plus judicieux et efficace. Cela d'autant plus qu'en utilisant une mémoire particulière à double port, l'inconvénient majeur du goulet d'étranglement sur le bus commun disparait. Une mémoire à double port permet à deux processeurs des accès simultanés à la mémoire autant en écriture qu'en lecture tant que les adresses visées ne sont pas simultanément les mêmes. Cela en mettant à disposition deus séries de bus d'accès.

Les mémoires de type double port étant particulièrement onéreuses, la puce la plus simple parmi l'éventail disponible a été retenue. Le modèle IDT71V30L25TFG a en outre l'avantage d'être très rapide (25 ns de temps d'accès) et il est capable de générer un signal nommé *BUSY* qui indique au processeur s'il doit attendre qu'une adresse de la mémoire soit libérée avant qu'il puisse l'accéder. Son bus de données est de 8 bits de large et sa taille de 1kByte. C'est donc par cette mémoire que transiteront tous les échanges entre les deux processeurs: données et messages. La Figure 5 montre la présence de deux bus de contrôle (*OE, CE, R/W, BUSY et INT*), deux bus d'adressage sur 10 bits (*A0 - A9*) et deux bus de données sur 8 bits (*I/O0 - I/O7*). Chaque processeur ayant ses propres bus, aucun protocole gérant l'accès aux bus ne doit être implémenté.

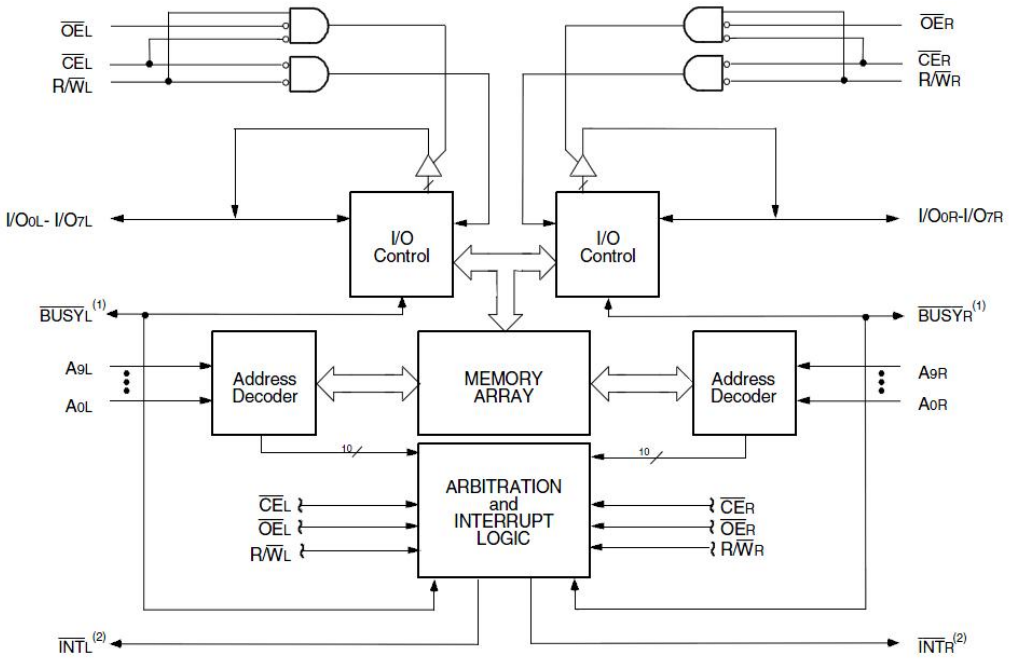


Figure 5 : Schéma bloc de la mémoire double port

3.2.2 Processeur et schéma bloc avancé de la carte

Cortex est la dernière évolution d'architecture 32 bit conçue par l'entreprise ARM, elle est partagée en trois familles : A, R et M. Les Cortex-A délivrent de très hautes performances (jusqu'à 2500 **MIPS**), les R sont compatibles au niveau du code avec les anciennes architectures ARM et les M sont spécialisés dans le temps réel et l'efficacité énergétique. Ils peuvent fournir jusqu'à 125 MIPS.

Le schéma bloc d'un cœur Cortex-M3 est visible en Figure 6. Ses particularités sont, entre autres, un contrôleur d'**interruptions** (NVIC) très performant et déterministe, une architecture interne multi-bus (Bus Matrix) et des interfaces de debuggage complexes comme le tracage (ETM).

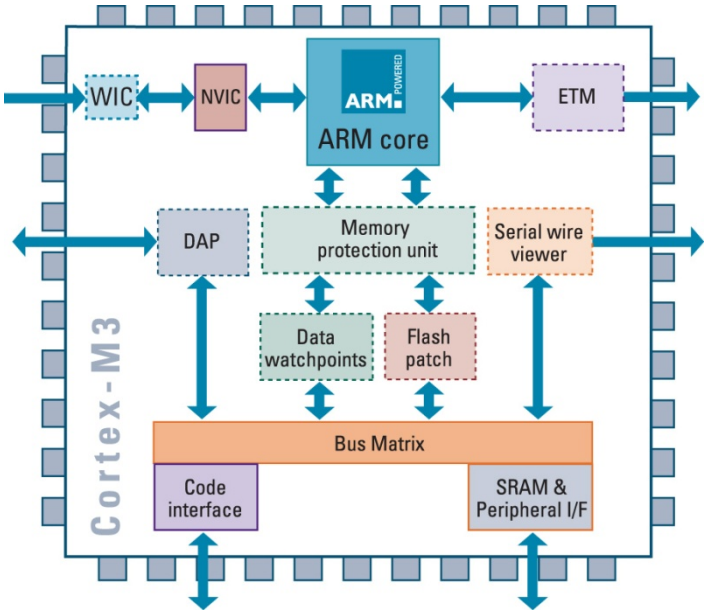


Figure 6 : Cœur Cortex-M3

ARM ne fabrique pas de circuits intégrés, elle développe des architectures de cœurs qui sont ensuite vendues sous licence à des producteurs de **System-on-Chip** (SoC) comme Atmel, STm ou Samsung. A partir de l'architecture du cœur, des composants comme de la mémoire (ram et flash) ou des périphériques sont ajoutés selon les besoins et intégrés dans une même puce. Il existe donc une multitude de puces à base de cœurs Cortex-M3.

Le département d'Infotronique de l'école collabore avec l'HEPIA (Haute École du Paysage, d'Ingénierie et d'Architecture de Genève) sur un projet (NTRT) utilisant déjà un processeur STm Cortex-M3, cela a directement influencé le choix de la marque du processeur choisi. De plus, leur gamme de SoC à base de Cortex-M3 est très étendue.

Les éléments cruciaux à évaluer lors du choix d'un microcontrôleur sont la quantité de mémoire vive (SRAM) et morte (FLASH) et les périphériques intégrés à la puce.

- La volonté de porter un OS temps réel sur la carte implique que la quantité de mémoire vive disponible dans le processeur soit suffisante. En annexe II, un tableau permet d'estimer la quantité de mémoire vive nécessaire à RTEMS en fonction du nombre d'éléments utilisés (threads, **sémaphores** ou **stacks**). Celui-ci montre qu'un minimum de 12kBytes de ram est nécessaire.

- La gestion de la mémoire externe est grandement simplifiée si le processeur choisi contient un contrôleur de mémoires externes dédié. Aussi, certains modèles possèdent un contrôleur USB intégré, ce qui éviterait de devoir ajouter des puces supplémentaires pour implémenter cette fonctionnalité.
- Le prix de ces puces oscillant entre 10 et 25 francs, ce facteur n'est pas limitant dans le cadre de ce projet.

Le processeur STM32F103ZET6 a donc été choisi. Il est disponible dans le catalogue Farnell et c'est exactement le même utilisé pour NTRT. Ce microcontrôleur possède 64kBytes de RAM, 512kBytes de FLASH, un contrôleur de mémoires externes (Flexible Static Memory Controller) et gère l'USB. Sa fréquence d'horloge peut aller jusqu'à 72 Mhz et ses performances sont de 1,25 MIPS/Mhz (90 MIPS max). Elément remarquable, ce processeur pourtant classé dans le haut de la gamme ne dispose d'aucune cache.

Les deux éléments principaux du système à développer (architecture mémoire et processeur) étant maintenant définis, il est utile de montrer un schéma bloc (Figure 7) approfondi mettant en évidence les interactions entre chaque composant.

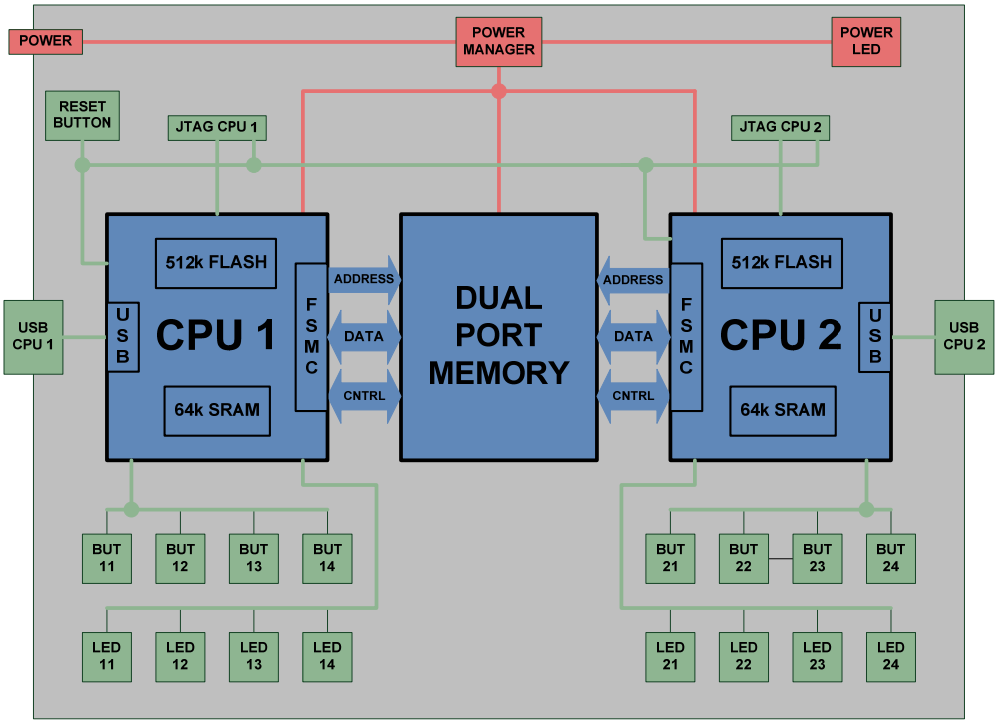


Figure 7 : Schéma bloc détaillé

Il est intéressant de noter que la présence de mémoire intégrée dans les processeurs modifie l'architecture mémoire initiale décidée. Cette architecture est donc un hybride d'architecture distribuée et partagée, permettant de profiter de la vitesse d'accès rapide du processeur à sa mémoire interne et de la simplicité d'échange d'information entre les processeurs qu'offre la mémoire commune.

3.2.3 Liaison processeurs – mémoire

Maintenant que les composants principaux de la carte sont décidés, la réalisation de la schématique électrique peut commencer. Ce chapitre décrit le routage des signaux entre la mémoire et les processeurs.

Les pins de la puce mémoire sont placés de manière symétrique. Seul le branchement d'un des processeurs aux signaux de la mémoire avec l'exposant $_L$ pour left est expliqué, le branchement de l'autre se fait de manière similaire sur les signaux avec l'exposant $_R$ pour right.

Le contrôleur de mémoires externes présent dans le processeur, appelé FSMC, a des pins définis qui lui sont attribués, il y a donc peu de choix quand au branchement de la mémoire. En annexe III, un tableau extrait du datasheet STM32F103ZET6.pdf (*cd-rom /hardware/datasheets_composants/processeur*) explique les fonctions par défaut et alternatives de tous les pins. En annexe IV se trouve le pining de la mémoire.

Le choix de branchement du signal de **chipEnable** ($\overline{CE}_L$) de la mémoire mérite une explication. Les mémoires externes sont accessibles par le FSMC via 4 espaces d'adressages appelés ici banques (de 1 à 4). Seule la banque 1 (0x6000 0000 à 0x6FFF FFFF) peut être utilisée pour gérer une SRAM externe, les trois autres étant réservées aux mémoires flash NAND ou à un slot PC Card. Cela est visible à la Figure 8 tirée du datasheet STM32F10xx_advanced_datasheet.pdf (*cd-rom /hardware/datasheets_composants/processeur*).

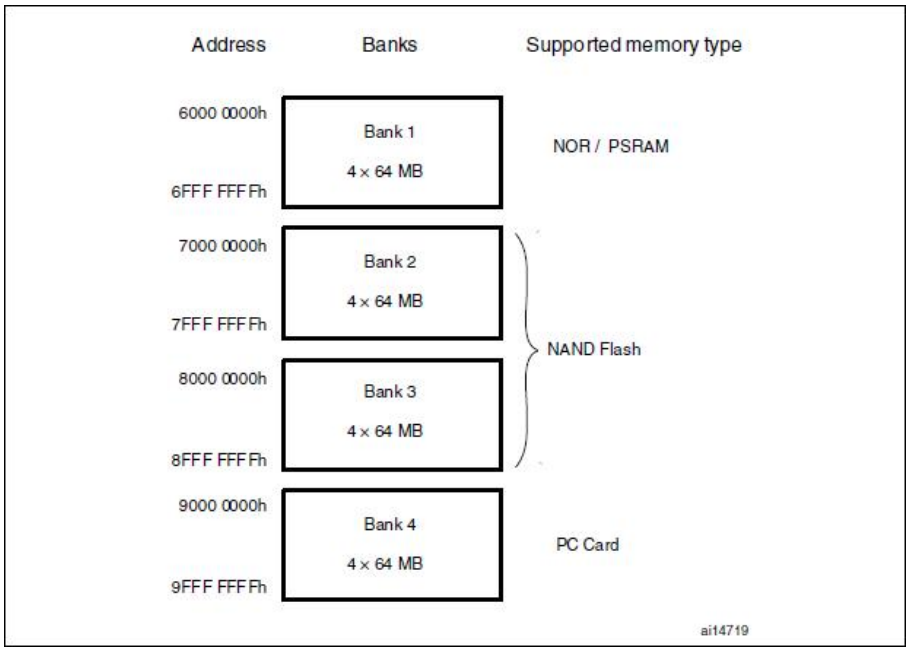


Figure 8 : Banques du FSMC

La banque 1 est elle-même partagée en 4 parties avec 4 signaux chipEnable distincts (*FSMC_NE1 à 4*). Pour que la mémoire externe soit accessible par les adresses 0x6000 0000 à 0x6000 0400 (1024 adresses) qui sont dans la première partie, il faut que le pin *PD7 – FSMC_NE1* soit branché sur le pin $\overline{CE}_L$ de la mémoire.

Les signaux d'adresse *A0* $_L$ à *A9* $_L$ doivent être branchés sur les pins *FSMC_A0 à A9* respectivement, comme expliqué dans le tableau 86 du datasheet STM32F10xx_advanced_datasheet.pdf.

Il est à noter que le pin *FSMC_WAIT* qui est sensé gérer le signal $\overline{BUSY}_L$ de la mémoire n'est utilisé que dans le cas d'une **mémoire synchrone** pour des transactions mémoire en **burst mode** selon la documentation (datasheet STM32F10xx_advanced_datasheet.pdf pages 433 à 440). Or, la mémoire utilisée dans ce projet est asynchrone et ne supporte pas de burst mode… La supposition a été faite que la documentation à ce sujet était incomplète ou erronée et que le FSMC était tout de même capable de gérer ce signal pour une mémoire asynchrone sans burst mode comme celle choisie ici. Cette supposition s'est révélée être correcte dans la pratique. Le signal *FSMC_WAIT* est actif lorsque la mémoire détecte un accès simultané des deux processeurs à la même adresse mémoire. Son effet est de faire ajouter au FSMC des **waitstates** pendant une transaction mémoire tant qu'il est actif, afin d'attendre la libération de l'emplacement mémoire.

Le pin $\overline{INT}_L$ (signal d'interruption de la mémoire), bien que non nécessaire, a été routé vers le pin *PA4* du processeur dans le cas où une application particulière pourrait en faire usage. Le choix de branchement sur le pin *PA4* est expliqué dans le chapitre 3.2.6.1 Boutons et interruptions externes, où le problème des interruptions externes est traité.

Voici un tableau en Figure 9 résumant la connexion d'un processeur à la mémoire.

PIN mémoire	PIN processeur		PIN mémoire	PIN processeur
$\overline{OE}_L$	PD4 - FSMC_NOE		A_{0L}	PF0 - FSMC_A0
$\overline{CE}_L$	PD7 - FSMC_NE1		A_{1L}	PF1 - FSMC_A1
$R/\overline{W}_L$	PD5 - FSMC_NWE		A_{2L}	PF2 - FSMC_A2
$\overline{BUSY}_L$	PD6 - FSMC_NWAIT		A_{3L}	PF3 - FSMC_A3
$\overline{INT}_L$	PA4		A_{4L}	PF4 - FSMC_A4
I/O_{0L}	PD14 - FSMC_D0		A_{5L}	PF5 - FSMC_A5
I/O_{1L}	PD15 - FSMC_D1		A_{6L}	PF12 - FSMC_A6
I/O_{2L}	PD0 - FSMC_D2		A_{7L}	PF13 - FSMC_A7
I/O_{3L}	PD1 - FSMC_D3		A_{8L}	PF14 - FSMC_A8
I/O_{4L}	PE7 - FSMC_D4		A_{9L}	PF15 - FSMC_A9
I/O_{5L}	PE8 - FSMC_D5			
I/O_{6L}	PE9 - FSMC_D6			
I/O_{7L}	PE10 - FSMC_D7			

Figure 9 : Récapitulatif pining mémoire

Les pages de la schématiques concernées pas cette connexion se trouvent en annexes VIII, X et XI.

3.2.4 JTAG et sérialisation de la chaine

Le JTAG (Joint Test Action Group) est le nom de la norme IEEE 1149.1 intitulée « Standard Test Access Port and Boundary-Scan Architecture ». Le document JTAGspec.pdf spécifiant cette norme se trouve sur le cd-rom dans /hardware/JTAG.

Initialement, le Boundary Scan (littéralement, scrutation des frontières) était destiné au test des court-circuits et de la continuité entre puces compatibles. Connaissant le schéma d'une carte ou d'une puce, un ensemble de signaux logiques (appelé vecteur de test) est appliqué sur les broches d'entrée de certains composants et les niveaux logiques sur les broches de sortie sont relevés pour s'assurer qu'ils correspondent aux valeurs attendues. Le bon fonctionnement d'une puce ou d'un PCB peut ainsi être testé et validé. De même, il est possible de tester des mémoires en écrivant puis relisant des valeurs de test. De cette manière, le Boundary Scan permet de programmer des mémoires non-volatiles comme des EEPROM ou des FLASH et surtout d'effectuer du debug d'application (break points, lecture du contenu des registres et de la mémoire) en cours d'exécution sur une puce grâce à la technique ICD (In-Circuit Debugger). C'est donc grâce à ce port que les applications développées seront chargées dans les processeurs et debuggées.

Le bus JTAG est un bus série synchrone composée de 5 signaux :

- TMS (*Test Mode Select*) : Signal de contrôle de la machine d'état TAP
- TCK (*Test Clock*) : Horloge
- TDI (*Test Data Input*) : Entrée des données,
- TDO (*Test Data Output*) : Sortie des données,
- TRST* (*Test Reset*) : Réinitialisation asynchrone du TAP actif à l'état bas. *optionnel

Dans le cadre de ce travail de diplôme, ces signaux ont été renommés JTMS, JTCK, JTDO, JTDI et NJTRST.

Le TAP (Test Access Port), commandé par le signal JTMS, contrôle le comportement du protocole JTAG. Sa machine d'état, implémentée à l'intérieur de la puce du processeur, est consultable en annexe V. A chaque flanc montant du signal JTCK, la valeur de JTMS déclenche la transition à l'état suivant. Par exemple, le reset synchrone du TAP s'effectue avec une séquence de cinq '1' sur la ligne JTMS, cela depuis n'importe quel état. A chaque flanc montant de JTCK, la valeur présente sur JTDI est « injectée » dans la chaine et la valeur « poussée » hors de la chaine est lue sur JTDO au flanc descendant successif de JTCK.

Selon les spécifications de la norme JTAG, les signaux JTMS, JTDI et JTRST doivent être munis d'une résistance pull-up. La spécification JTAG n'impose rien pour le signal JTCK cependant la norme en vigueur pour les processeurs STm voudrait qu'une résistance pull-down soit utilisée. Cependant, une pull-up est implémentée sur ce pin. Elle produit le même effet qu'une pull-down mais a l'avantage d'annuler le courant de fuite dans le transistor d'entrée du pin. Toutes ces pull-up sont implémentées avec des résistances de 10k Ohm.

Pour des raisons de « sécurité », il est initialement prévu d'avoir un port JTAG indépendant par processeur pour garantir que la programmation et le debuggage sur chacun des processeur pourront s'effectuer sans problèmes. Cependant cette configuration a le désavantage de devoir utiliser deux **dongles** JTAG et deux instances de l'environnement de développement. De plus, il sera impossible de stopper l'exécution de code sur les deux processeurs simultanément, ce qui peut être gênant. Afin de simplifier et d'améliorer ces tâches, une option matérielle de sérialisation la chaine JTAG doit être conçue dans le but de n'utiliser qu'un port JTAG.

La Figure 10 montre à gauche deux chaines JTAG parallèles et à droite une chaine unique sérialisée.

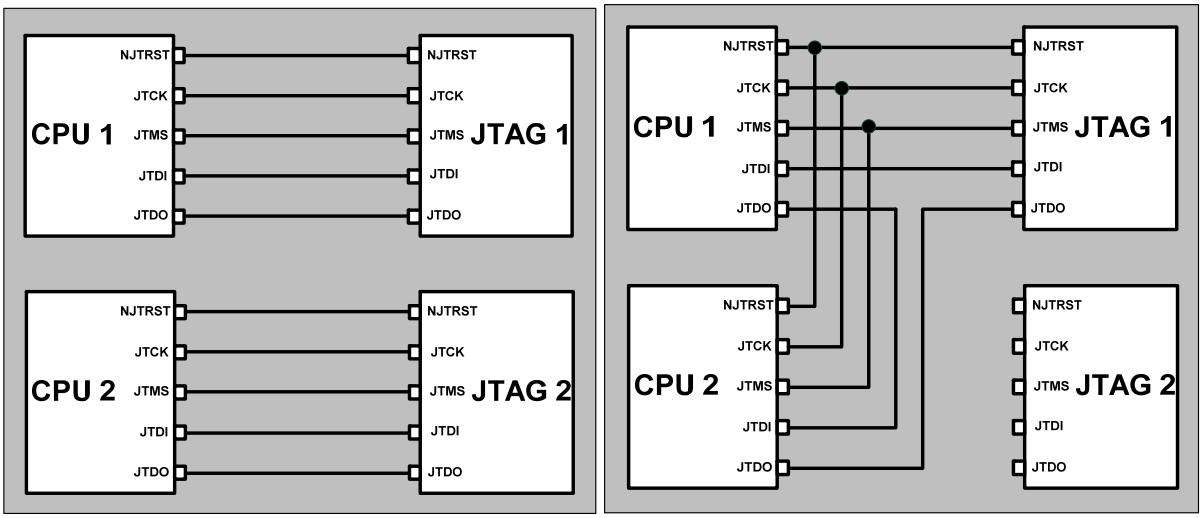


Figure 10 : Chaine JTAG parallèle et série

Dans le mode sérialisé, les signaux JTMS, JTCK et NJTRST sont transmis simultanément aux deux processeurs par le même port JTAG, la sérialisation concerne les signaux JTDI et JTDO. Le JTDO du premier processeur doit être connecté au JTDI du second et c'est le signal JTDO de ce dernier qui doit être connecté au port JTAG.

Des résistances 0 Ohm sont utilisées pour effectuer le passage d'un mode à l'autre. Bien que moins pratiques que des jumpers, les résistances ont l'avantage de moins perturber les signaux quand leur fréquence est élevée. Pour un processeur cadencé à 72 Mhz, la fréquence recommandée pour l'horloge de son port JTAG est de 8Mhz.

Le montage montré à la Figure 11 permet cette modification.

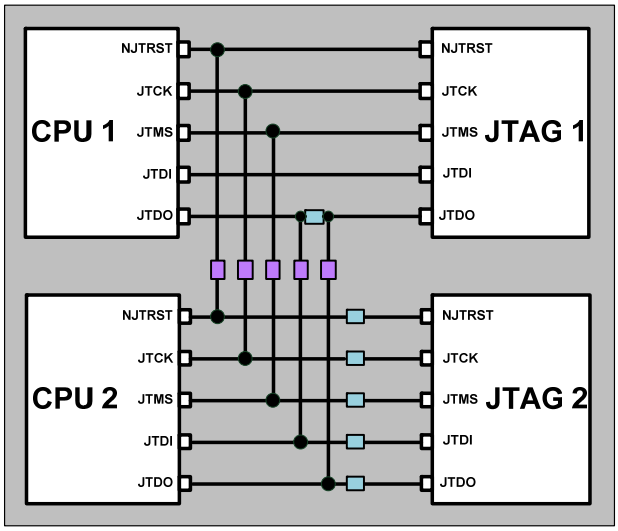


Figure 11 : Possibilité de modularité JTAG

Lorsque les résistances en bleu ciel sont soudées et que les emplacements des résistances mauves sont dépeuplés, les deux chaines JTAG sont en parallèle. Si c'est l'inverse, la chaine JTAG est sérialisée et tout transite donc par un seul dongle.

Les pins du processeur affectés au JTAG sont imposés comme suit :

- JTMS : *PA13 JTMS-SWDIO*
- JTCK : *PA14 JTCK-SWCLK*
- JTDI : *PA15 JTDI*
- JTDO : *PB3 JTDO*
- NJTRST : *PB4 NJTRST*

Les connecteurs JTAG montés sur la carte sont imposé par la norme en vigueur dans le département d'Infotronique de la HES-SO de Sion, leur référence est 98414-G06-10ULF et ils ont disponibles chez Distrelec sous le numéro d'article suivant : 11 67 97. Les deux utilisés pour la carte bicortex ont été pris du stock de l'école.

La schématique électrique des ports JTAG est en annexe XII et XIII.

3.2.5 Fonctionnalités du reset

Un signal de reset est nécessaire pour effectuer une remise à zéro du système pendant que celui-ci est alimenté. C'est un signal actif à l'état bas qui doit d'une part reseter le processeur et d'autre part initialiser la machine d'état TAP gérant le JTAG. La décision a été prise que ce signal affecterait les deux processeurs afin de les reseter simultanément. Ce signal provient d'un bouton exclusivement dédié à cette fonction.

De plus, chacun des ports JTAG peut générer un signal devant uniquement reseter le processeur (ce n'est pas le même signal que NJTRST qui resete uniquement le TAP) via un signal en plus des cinq vus dans le chapitre précédent.

Une logique combinatoire simple doit donc être implémentée afin de distribuer ces sources de reset correctement.

Tous ces signaux sont actifs à l'état bas, il faut donc prendre garde à utiliser des portes logiques ET et non pas OU pour obtenir un signal actif lorsqu'une de deux entrées est active comme montré par la table de vérité suivante (Figure 12).

input1		input2		or		and	
actif	0	actif	0	actif	0	actif	0
actif	0	inactif	1	inactif	1	actif	0
inactif	1	actif	0	inactif	1	actif	0
inactif	1	inactif	1	inactif	1	inactif	1

Figure 12 : Logique combinatoire inverse

Les portes logiques ET choisies NC7S08M5X sont de simples tiny logic nécessitant une alimentation à 3.3V.

Les deux signaux de reset processeur provenant des ports JTAG ($\overline{CPU1\_JTAG\_RESET}$ et $\overline{CPU2\_JTAG\_RESET}$) ont dans un premier temps été couplés pour produire un signal $\overline{JTAG\_RESET}$ commun. Ce dernier est ensuite couplé avec le signal $\overline{BUTTON\_RESET}$ provenant du bouton de reset. Le signal résultant est câblé directement vers le pin NRST de chacun des processeurs. La Figure 13 illustre ce montage.

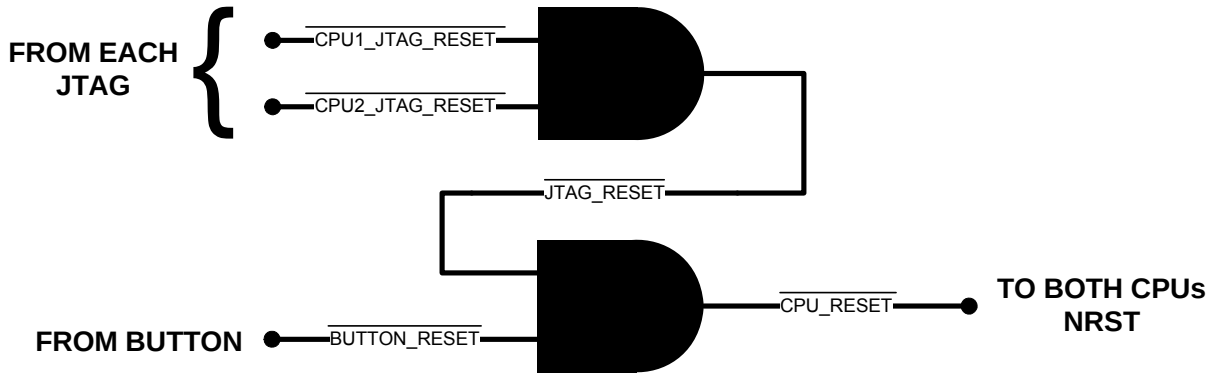


Figure 13 : Logique pour reset processeur

Le reset du TAP provenant des connecteurs JTAG doit initialiser uniquement le processeur concerné (si les deux chaines JTAG sont en parallèle), contrairement au reset provenant du bouton. Ils sont donc couplés indépendamment pour chacun des processeurs, produisant les signaux $\overline{CPU1\_NJTRST}$ et $\overline{CPU2\_NJTRST}$. Ce reset des TAP est soumis à la même option de sérialisation que les autres signaux du JTAG discutée au chapitre précédent. C'est la raison pour laquelle les résistances 0 Ohm ont été introduites sur les lignes comme le montre la Figure 14.

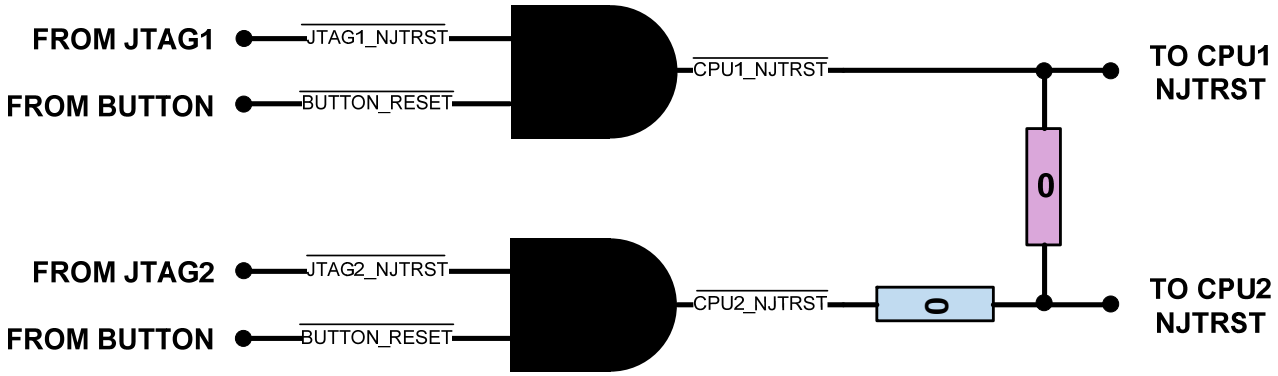


Figure 14 : Logique reset tap

Le signal provenant du bouton servant d'entrée à des portes logiques, il ne doit en aucun cas être laissé flottant, d'où l'adjonction d'une résistance pull-up sur sa ligne.

La schématique électrique de la logique de reset est disponible en annexe XIII.

3.2.6 Entrées-sorties

3.2.6.1 BOUTONS ET INTERRUPTIONS EXTERNES

Il a été décidé que le signal des boutons est actif à l'état bas. Bien que le signal des boutons soit flottant lorsqu'ils ne sont pas appuyés, l'ajout de résistances pull-up n'est pas nécessaire car elles sont incorporées aux pins du processeur et activables via un registre de configuration.

Dans la schématique initiale, les boutons avaient été routés vers les pins *PB12 à 15*. Mais après une étude approfondie du contrôleur d'interruptions externes (EXTI) du processeur, la constatation a été faite que ces 4 pins ont une seule ligne d'interruption commune. Or, il est préférable que chacun des boutons dispose de sa propre ligne. C'est la raison pour laquelle les boutons sont finalement connectés sur les pins *PA0 à 3*. C'est cette même raison qui a conduit à router la ligne d'interruption de la mémoire sur le pin *PA4*.

Le choix des boutons a été rapide et conditionné par la présence dans le stock de l'école du modèle ITT KSAOM210 dont les caractéristiques satisfont aux besoins de ce projet.

La schématique électrique des boutons se trouve en annexe XIV.

3.2.6.2 LEDS

Le moyen le plus simple pour donner un feedback visuel sur ce que fait le processeur en exécutant une application est encore d'allumer ou d'éteindre des leds. L'utilisation de 4 leds par processeur offre un éventail de possibilités suffisant dans le cadre de ce projet. Les leds choisies sont des KP3216 SGD émettant une lumière verte.

De plus, il est pratique de savoir instantanément si la carte est sous tension ou non, une led prévue à cet effet est allumée lorsque la carte est alimentée. Cette led étant presque constamment allumée, le choix de ce composant a été conditionné par la consommation électrique. La led choisie (Vishay TLMO 1000) consomme 2 mA soit 10x moins qu'une led conventionnelle, elle émet une lumière orange.

Pour implémenter une led, il suffit de la brancher à un pin de sortie du processeur avec une résistance en série. Cette résistance doit être dimensionnée de sorte à limiter le courant dans la led à sa valeur nominale. La valeur de la résistance dépend donc de la chute de tension aux bornes de la led *Uf*, de la tension délivrée par le pin pour allumer la led *Vcc* et du courant nominal souhaité dans la led *In*. Voici l'équation, tirée directement de la loi d'Ohm, permettant de dimensionner cette résistance :

$$R_{led} = \frac{Vcc - Uf}{In}$$

Pour la led d'alimentation, *avec Vcc* = 3.3V, *Uf* = 1.8V et *In* = 2mA, la valeur de la résistance est de 750 Ohm. Pour les leds de feedback, *Vcc* est inchangé, *Uf* = 2,2V et *In* = 20mA et la résistance doit être de 55 Ohm. Cette valeur n'étant pas disponible dans les sets de résistances standard, une résistance de 56 Ohm a été choisie.

Il est important de ne pas utiliser de leds qui nécessitent plus que 25mA, les pins du processeur ne pouvant pas fournir plus de courant.

Le choix des pins n'étant pas critique pour les leds, le choix s'est porté arbitrairement sur les pins *PC0 à PC3*. La schématique électrique des leds se trouve aussi en annexe XIV.

3.2.6.3 PORTS USB

L'implémentation d'une interface USB sur la carte accroit ses possibilités d'utilisation. La présence d'un contrôleur USB interne dans le processeur requiert donc uniquement l'implémentation d'un connecteur USB de type A afin de brancher des périphériques à la carte. Tous les condensateurs ainsi que les résistances visibles à la Figure 15 servent à minimiser les bruits électromagnétiques et à adapter l'impédance des lignes afin de garantir un débit maximal. Les résistances qui font office de pull-up/down n'ont pas été montées, les emplacements ont justes été introduits pour pouvoir améliorer la qualité du signal le cas échéant.

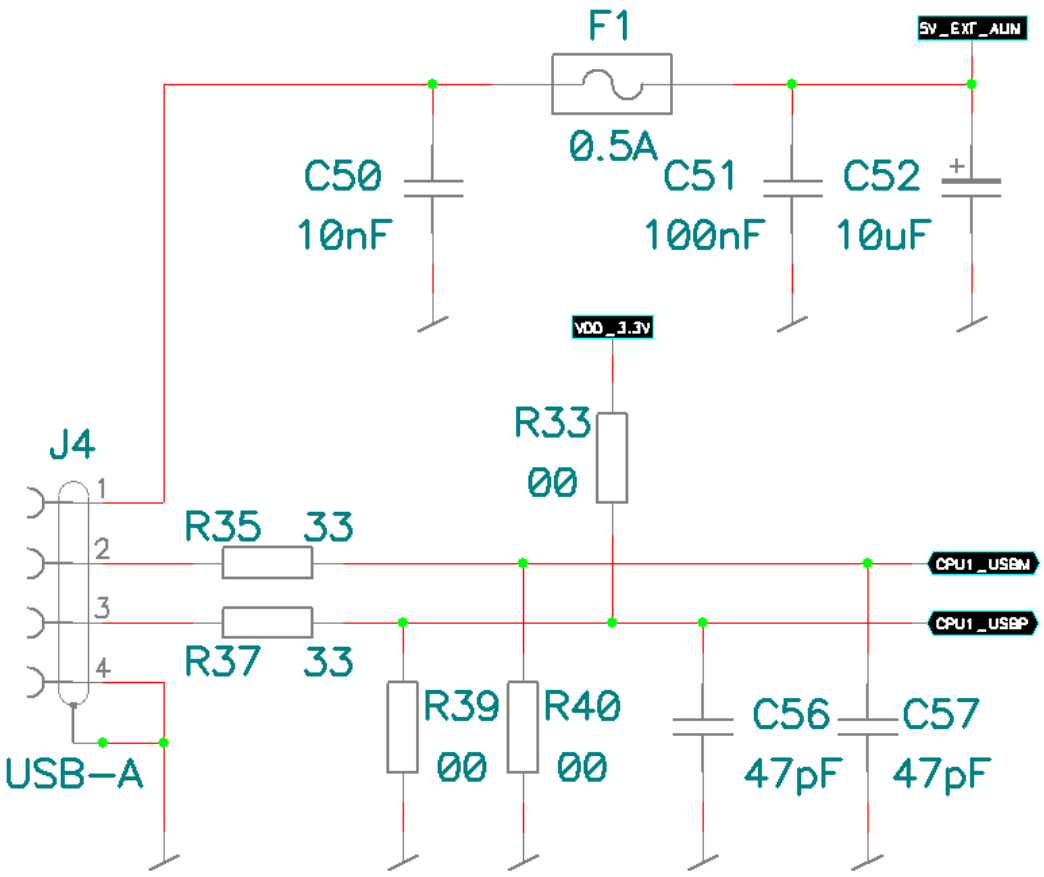


Figure 15 : Schématique Usb

La norme USB requiert que le courant maximal fourni à un périphérique ne dépasse pas les 500mA pour une tension de 5V. L'énergie est donc directement tirée des 5V de l'alimentation externe. Le fusible sert à limiter le courant et à protéger la carte dans le cas de la connexion d'un périphérique défectueux.

Une connexion USB sérielle nécessite 4 « fils » : une alimentation à 5V, une masse, et une paire différentielle (D+, D-) transportant l'information. Les pins du processeur gérant l'USB sont *PA11 – USBDM* pour D- et *PA12 - USBDP* pour D+.

Les connecteurs choisis sont les LUMBERG 2410 02.

La schématique électrique de ces ports USB est reprise de celle de la carte PICEBS3 réalisée au sein du département d'Infotronique. Elle est consultable en annexe XVI.

3.2.7 Alimentation

3.2.7.1 STEP-DOWN5V -> 3.3V

La décision d'alimenter la carte à une tension de 5V est due au fait que la plupart des alimentations de laboratoire non réglables fournissent cette tension par défaut. La majorité des éléments de la carte devant être alimentés à 3.3V, la réalisation d'un circuit de gestion de l'alimentation est indispensable.

Ce circuit doit convertir la tension d'entrée de 5V en une tension de 3.3V et il doit pouvoir fournir assez de courant pour satisfaire aux besoins de la carte. Une estimation de la consommation maximale de la carte doit donc être réalisée. Celle-ci est visible en annexe VI. A noter que les 500mA maximaux que peuvent fournir les ports USB ne proviennent pas du step-down mais directement de l'alimentation externe.

Le composant choisi pour gérer l'alimentation est donc un circuit intégré step-down Texas Instruments TPS62056DGSG4 acceptant des tensions d'entrée de 2,7 à 10V et fournissant à la sortie une tension de 3.3V. Il est capable de fournir un courant maximal de 800mA satisfaisant la consommation maximale estimée de 712mA.

Ce circuit, comportant une boucle de feedback pour s'autoréguler, est particulièrement sensibles aux perturbations électromagnétiques, ils nécessitent donc des composants particuliers afin de fonctionner correctement. Deux condensateurs de découplage **Low ESR**, un de 10uF (TPSB106K016R0500) à l'entrée et un de 22uF (NOSB226M004R0600) à la sortie doivent être utilisés, ainsi qu'une inductance de 15mH (ELL6UH150M) à faible résistance interne et supportant le courant maximal que le circuit peut lui injecter. Il est à noter que ce courant ne correspond pas au courant maximal fourni pouvant être fourni en continu par le step-down de 800mA. Il est estimé a 1,4 A dans le datasheet du composant pendant de très courts instants.

Ce step-down offre des fonctionnalités annexes au travers des pins LBO/FC, PG, LBI/ILIM et SYNC inutiles dans le cadre de ce projet. La documentation explique que ces pins doivent être connectés à la masse s'ils ne sont pas utilisés.

La schématique électrique réalisée est montrée en Figure 16

.

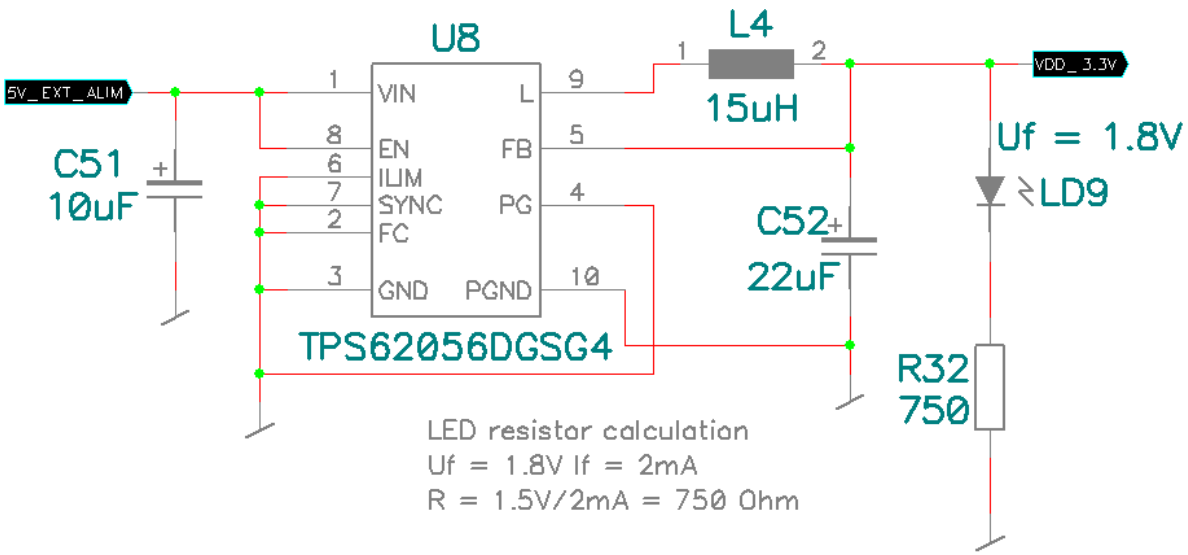


Figure 16 : Schématique step-down

3.2.7.2 PARTICULARITÉS DU DÉCOUPLAGE

L'alimentation d'une puce doit toujours être filtrée afin de garantir son bon fonctionnement. Cela est généralement fait en plaçant une batterie de condensateurs de différentes valeurs en parallèle entre les pins Vcc et GND du composant. La documentation du processeur STM32F103ZET6.pdf indique à la figure 12 qu'un seul condensateur de 100nH est suffisant pour chaque pin d'alimentation, sauf pour le pin Vcc3 qui nécessite un condensateur de 4.7uH en parallèle supplémentaire et pour le pin VDDA qui lui nécessite un condensateur de 1uF en parallèle avec un autre de 10nF.

Des **ferrites** (BLM18AG102SN1D) mises en série sur les lignes d'alimentation des processeurs et de la mémoire permettent d'obtenir un filtrage encore meilleur.

La schématique complète de l'alimentation et du découplage des processeurs et de la mémoire se trouve en annexes VII, IX, XI et XV.

3.2.8 Oscillateur 8MHz et quartz 32.768kHz

La note 2 du tableau 2 du datasheet STM32F103ZET6.pdf indique que l'utilisation de l'interface USB du processeur requiert la présence d'une source d'horloge externe qui peut être soit un quartz soit un oscillateur de fréquence comprise entre 4 et 25 Mhz. Il avait initialement été prévu d'utiliser un quartz indépendant par processeur. Cependant, les contraintes d'oscillation des quartz étant parfois difficiles à obtenir, le choix d'utiliser un oscillateur commun pour les deux processeurs a finalement été adopté, le bon fonctionnement d'un oscillateur étant garanti. Le composant choisi (XO91050UITA) oscille à une fréquence de 8Mhz. Il est utilisé pour générer la fréquence interne du processeur. Cet oscillateur ne nécessite qu'un condensateur de découplage pour être opérationnel.

Son routage est disponible en annexe VII.

Dans un système temps réel, la notion de temps est fondamentale, il est donc souhaitable d'implémenter une horloge temps réel pouvant calculer des durées avec précision. L'horloge interne basse fréquence LSI du processeur peut, d'une puce à une autre, osciller de 30 à 60 kHz (!) et nécessite un timer interne du processeur (TIM5) pour être régulée à une précision acceptable par du software à implémenter. Cette option a été jugée trop laborieuse.

L'utilisation d'une seconde source d'horloge externe cadencée à 32,768 kHz, notamment utilisée dans l'horlogerie, apporte un gain en précision appréciable comparé à la source interne du processeur. Cette fois, le choix d'utiliser deux quartzs indépendants (CC7V-T1A 32.768Khz CL9) a été conservé. Le chapitre 5.3.6 page 54 du datasheet STM32F103ZET6.pdf décrit les différentes normes pour l'implémentation d'un quartz externe. Les pages 57 et 58 parlent spécifiquement du quartz à 32,768 KHz. Les deux capacités présentes sur le montage typique (Figure 17) en ont été dimensionnées avec l'équation donnée :

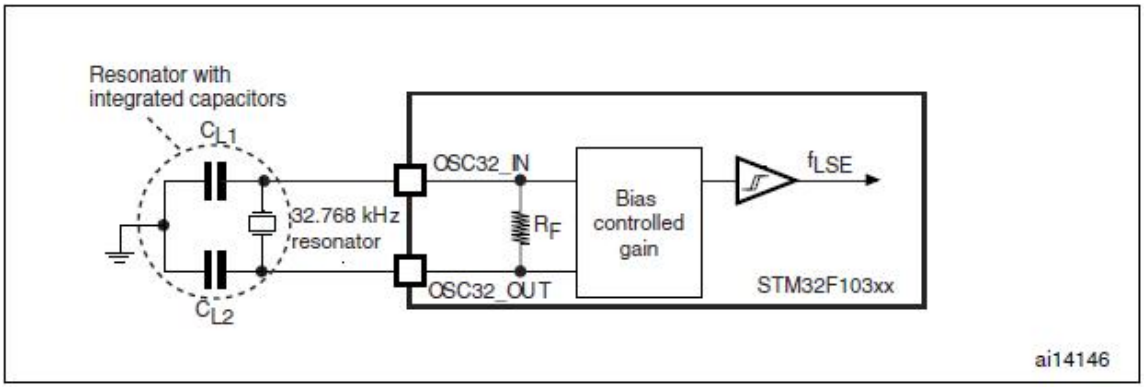


Figure 17 : Montage typique quartz 32,768kHz

$$\frac{Cl1 * Cl2}{Cl1 + Cl2} + Cstray = CL$$

- CL = 9pF selon le datasheet du quartz
- Cstray estimé à 4pf (en général compris entre 2 et 7 pF)
- Cl1 = Cl2 => Cl1 = 2*CL - 2*Cstray = 10pF

Cstray est la capacité parasite estimée des pins et de la ligne servant de connexion entre le quartz et le processeur.

La schématique de ces quartz est consultable aux annexes VIII et X.

3.2.9 Schématique, routage et production de la carte

La schématique électrique a été réalisée avec l'application P-CAD V2006 build 19.029589 qui est en général préinstallée sur les ordinateurs du département Infotronique. Il a été nécessaire de faire ajouter par l'atelier d'électronique de l'école certains composants, encore jamais utilisés, dans les librairies de P-CAD. La mémoire IDT71V30L25TFG se trouve dans la librairie MEMORY.LIB et le processeur STM32F103ZET6 dans PROCESS.LIB. Une fois la schématique validée par M.Pascal Sartoretti, le routage a été délégué aux ateliers susmentionnés.

Ce routage a été effectué sur un PCB quatre couches permettant une réduction de la taille de la carte ainsi qu'une diminution des problèmes liés aux interférences électromagnétiques. Les fichiers informatiques concernant le routage ainsi que les fichiers Gerber se trouvent dans le répertoire /hardware/schématique_routage du cd-rom. Les plans de routage des quatre couches se trouvent en annexes XVII, XVIII, XIV et XX.

L'école n'étant pas en mesure de réaliser des PCB 4 couches, la production a été réalisée par Eurocircuits. Produire deux PCB prototypes dans un délai de 5 jours a couté 165 €.

Les composants ont tous été commandés chez Farnell suisse, le bulletin de commande avec les numéros d'articles et les prix au moment de la commande se trouve en annexe XXI. Les deux connecteurs JTAG 98414-G06-10ULF , le connecteur 5V DC10B, la led basse-consommation TLMO 1000 , les leds de feedback KP3216 SGD et les boutons KSAOM 210 proviennent du stock de l'école.

Le soudage des composants sur la carte a été effectué par l'atelier d'électronique. Voici en Figure 18 une photo de la carte avec une mise en évidence des différents blocs.

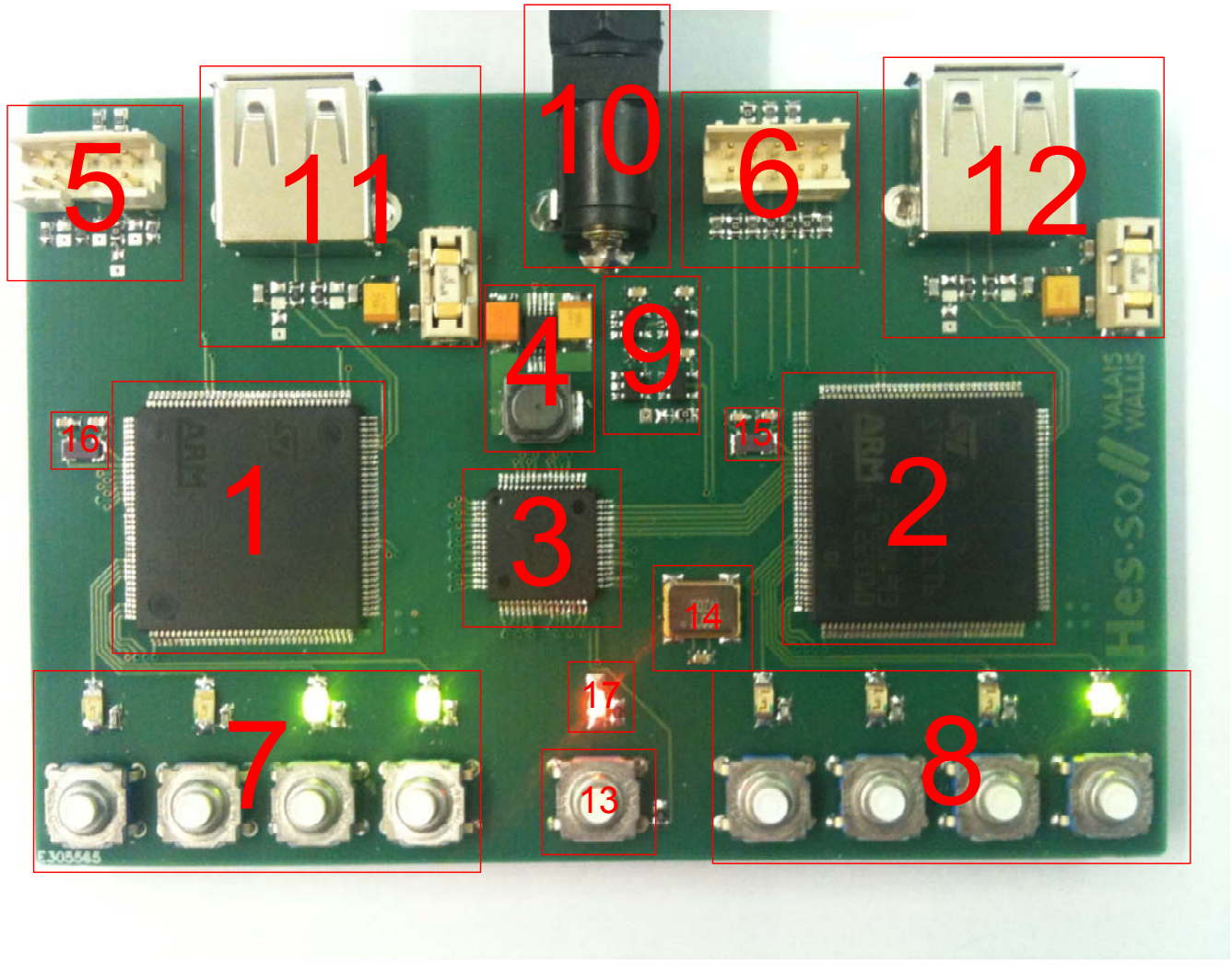


Figure 18 : Photo de la carte mère

- 1	: Processeur 1	Annexes VII et VIII	tous	
- 2	: Processeur 2	Annexes IX et X	tous	
- 3	: Mémoire Dual Port	Annexe XI	chapitre 3.2.3	
- 4	: Gestion de l'alimentation	Annexe XV	chapitre 3.2.7	
- 5	: Port Jtag processeur1	Annexe XII et XIII	chapitre 3.2.4	
- 6	: Port Jtag processeur2	Annexe XII et XIII	chapitre 3.2.4	
- 7	: Boutons et leds processeur 1	Annexe XIV	chapitre 3.2.6.1	3.2.6.2
- 8	: Boutons et leds processeur 2	Annexe XIV	chapitre 3.2.6.1	3.2.6.2
- 9	: Logique reset CPU et TAP	Annexe XIII	chapitre 3.2.5	
- 10	: Connecteur +5V	Annexe XV	chapitre 3.2.7	
- 11	: Connecteur USB processeur 1	AnnexeXVI	chapitre 3.2.6.3	
- 12	: Connecteur USB processeur 2	AnnexeXVI	chapitre 3.2.6.3	
- 13	: Bouton de Reset	Annexe XIII	chapitre 3.2.5	
- 14	: Oscillateur 8Mhz	Annexe VII	chapitre 3.2.8	
- 15	: Quartz 32.768kHz processeur 2	Annexe X	chapitre 3.2.8	
- 16	: Quartz 32.768kHz processeur 1	Annexe VIII	chapitre 3.2.8	
- 17	: Led alimentation	Annexe XV	chapitre 3.2.7	3.2.6.2

3.3 Tests hardware

Avant l'utilisation d'un circuit à peine sorti de production, une série de tests doit être effectuée sur la carte pour s'assurer d'une part de la sécurité du système pour l'utilisateur et d'autre part pour vérifier que la schématique conçue produit bien les fonctionnalités désirées.

3.3.1 Tests alimentation

3.3.1.1 VÉRIFICATION COURT-CIRCUITS

Le court-circuit est le pire ennemi d'un prototype. Une erreur de conception de la schématique ou une soudure défectueuse due à la taille minuscule des pins de certains composants entrainent parfois des courts-circuits. Alimenter la carte sans prendre de précautions peut l'endommager irrémédiablement.

Un test simple permet de vérifier s'il n'y a pas de court-circuit constant (indépendant des actions des processeurs). Il suffit de mesurer la résistance entre le GND et le VCC du connecteur d'alimentation 5V. La valeur mesurée sur cette carte est supérieure à 10Mohm. Elle garantit donc qu'aucun court-circuit n'est présent. Il est à noter que cette valeur n'est pas stable et croit avec le temps car l'ohm-mètre charge les condensateurs de la carte pendant la mesure, la faisant varier.

L'alimentation externe a ensuite été branchée. La précaution de limiter le courant à sa valeur minimale pour permettre à la carte de fonctionner (~250mA) a été prise car des courts-circuits peuvent encore survenir en fonction de l'état des pins du processeur ou de la mémoire.

Il sera constaté qu'aucun problème ne survient même lorsque toutes les fonctionnalités de la carte sont utilisées par les processeurs

3.3.1.2 VALIDATION TENSION

Il faut maintenant s'assurer que la tension délivrée par le step-down correspond bien à la valeur attendue de 3.3V. Une mesure à l'oscilloscope sur la borne positive du condensateur de sortie du step-down permet de mesurer non seulement la valeur de cette tension mais aussi son ondulation. Voici le résultat de cette mesure (Figure 19) :

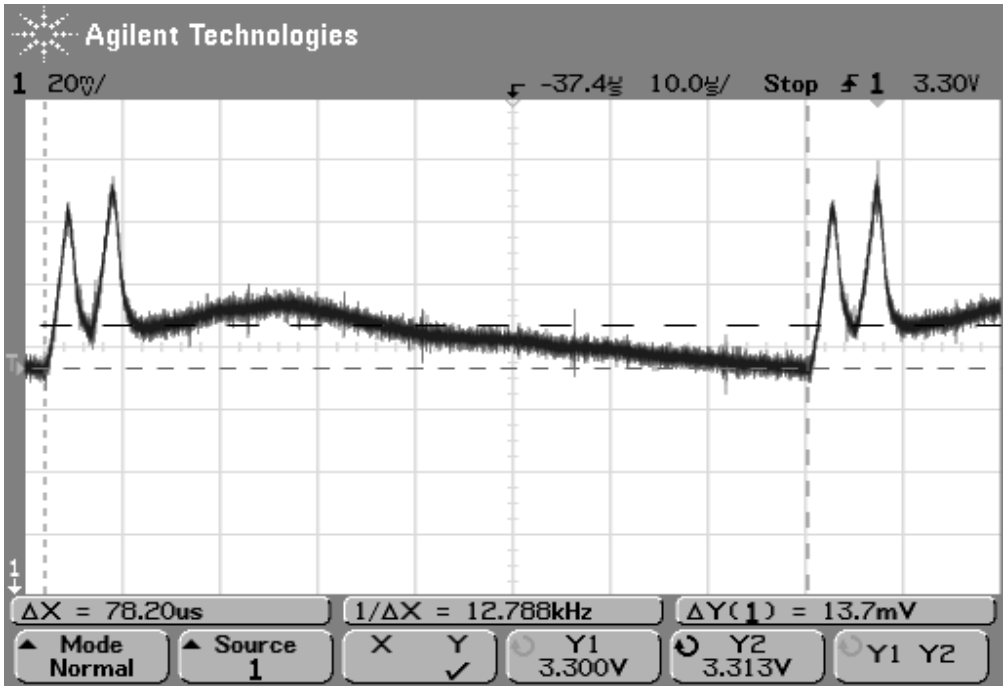


Figure 19 : Oscilloscope 3.3V

La valeur moyenne de ce signal est de 3.31V, ce qui est tout à fait satisfaisant Cependant une perturbation périodique anormale de fréquence proche des 13khz apparait également. Celle-ci entraine une ondulation du signal d'une valeur de 37mV. Une mesure supplémentaire similaire sur la tension de 5V fournie par l'alimentation externe a donc été réalisée. Le signal visualisé en Figure 20 montre une ondulation similaire et de même fréquence.

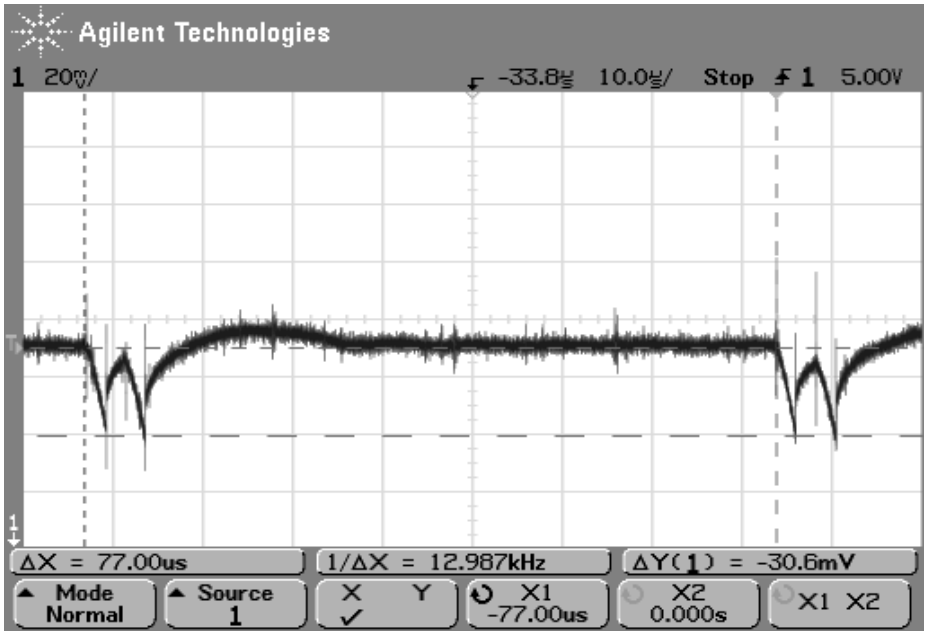


Figure 20 : Oscilloscope 5V

Cette perturbation est probablement due au fonctionnement du step-down ou à l'alimentation externe. Elle ne nuit cependant pas au bon fonctionnement du système, d'autant plus que les ferrites et les condensateurs de découplage sur chacun des composants réduisent considérablement les pics de cette ondulation. Cela est visible dans la Figure 21 qui montre le signal entrant directement sur un pin d'alimentation du processeur.

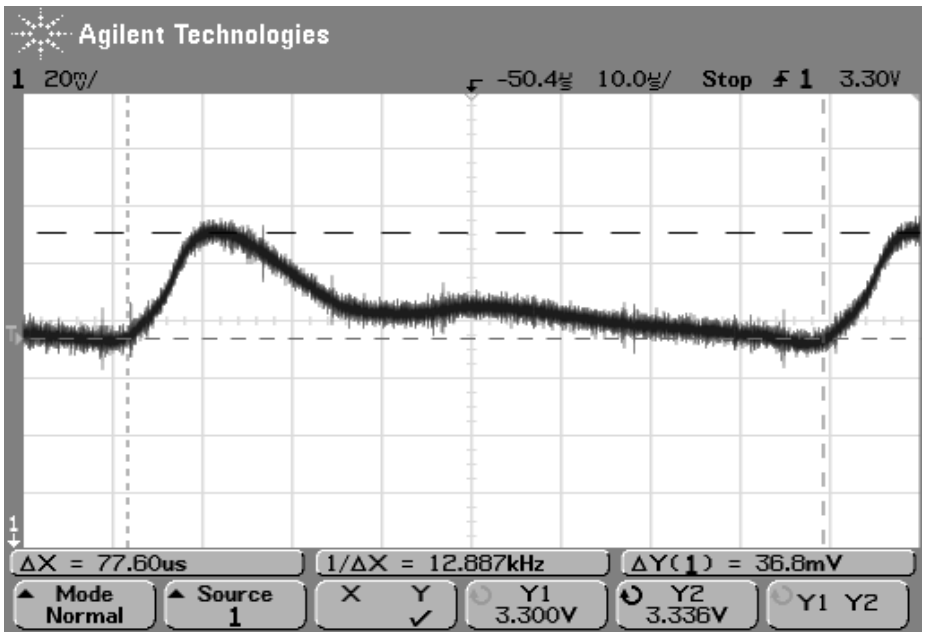


Figure 21 : Alimentation sur le pin VCC3 du processeur

Pour reproduire ces mesures, il est nécessaire de maintenir le bouton reset enfoncé tout au long de l'acquisition, l'activité des composants pouvant faire varier la charge sur le step-down et perturber la tension avec des champs électromagnétiques.

3.3.1.3 VÉRIFICATION DE TOUTES LES ALIMENTATIONS

L'alimentation de tous les circuits qui en nécessitent une a aussi été vérifiée. La liste de ces alimentations est la suivante :

3.3V

- 11 VCC de chaque processeur
- VDDA, VBAT et VREF+ de chaque processeur
- 2 VCC de la mémoire
- Pin 1 des connecteurs JTAG
- VCC des 4 AND tiny logic
- VCC et ENABLE de l'oscillateur 8Mhz

5V

- Pin 1 des connecteurs USB

Tous les composants sont dument alimentés.

3.3.2 Vérification JTAG

La schématique électrique concernant les résistances 0 Ohm pour la sérialisation de la chaine JTAG n'étant pas forcément très claire, le fonctionnement en mode parallèle des deux JTAG a été contrôlé à l'aide d'un ohm-mètre/beeper. Les deux ports JTAG sont bien routés comme prévu.

Cela a aussi permis de vérifier que le pining du connecteur JTAG est lui aussi correct.

3.3.3 Vérification oscillateurs

Le dernier test réalisé concerne l'oscillateur et les quartzs. La fréquence et les niveaux des signaux générés doivent correspondre à ce qui est attendu.

L'oscillateur à 8Mhz produit bien un signal carré avec un duty cycle de 50% et une fréquence de 8Mhz visibles à la Figure 22. De plus les niveaux mesurés sont bien dans les normes du processeur décrites dans le tableau 21 du datasheet SMT32F103ZET6.pdf : le niveau haut de 3,31V est bien compris entre 0,7xVCC et VCC et le niveau bas à 0V est bien entre 0.3VCC et GND.

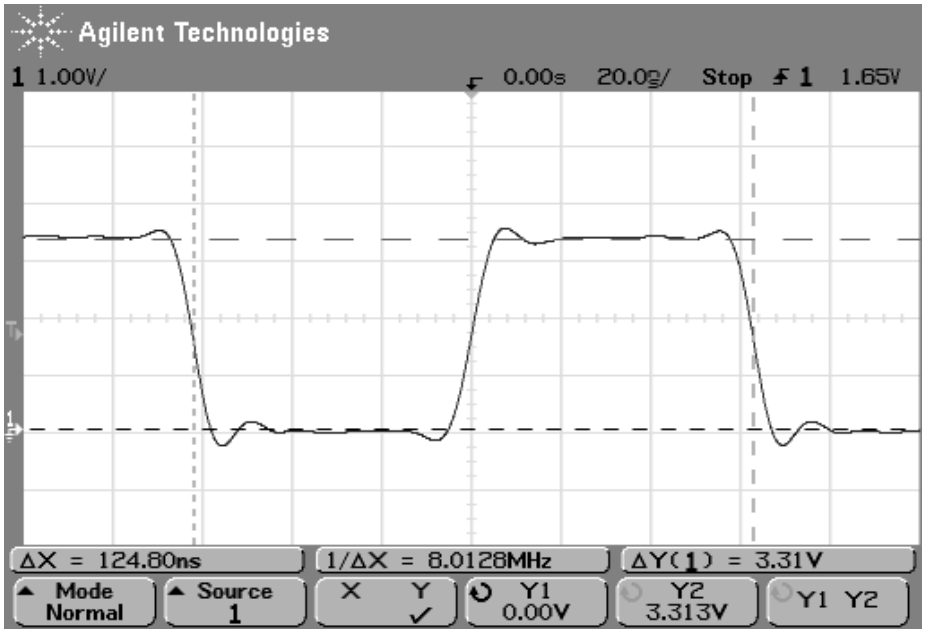


Figure 22 : Oscilloscope oscillateur 8Mhz

Les quartzs à 32'768 kHz nécessitent une configuration software de la carte pour être excités, ces tests ont donc été effectués dans un second temps. Les résultats sont cependant exposés ici même (Figure 23). Le gain de ce signal étant automatiquement adapté par le processeur, l'amplitude de ce signal n'est pas importante.

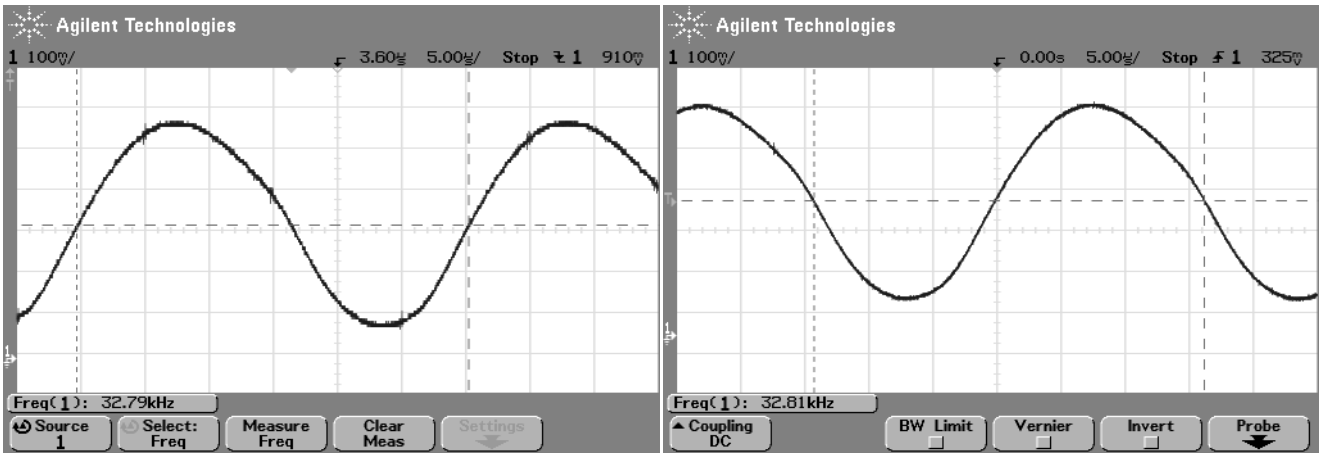


Figure 23 : Oscilloscope des quartzs 32,768 kHz

4 Développement software

Ce chapitre explique tous ce qui concerne les aspects logiciels de ce projet. Notamment la mise en place d'un environnement de développement et d'une toolchain compatible avec le processeur choisi. Une brève explication du fonctionnement du remote debugging sera donnée. La configuration des registres gérant les périphériques du processeur est abordée. Une partie sera aussi dédiée à la description de l'application de démonstration. Finalement, le travail effectué pour le portage sur le système de l'OS RTEMS sera expliqué.

4.1 Spécifications logicielles

Certains aspects logiciels de ce projet ont aussi été spécifiés :

- Eclipse a été choisi comme environnement de développement.
- la toolchain doit permettre de compiler et de télécharger une application sur chacun des processeurs. Elle doit aussi permettre de debugger le code à l'aide de breakpoints et d'un feedback sur le contenu des variables et de la mémoire.
- Le programme de démonstration devra mettre en évidence le gain en performances apporté par le second processeur.
- L'OS porté sera de préférence RTEMS à cause de sa certification aérospatiale. L'option a été laissée de porter microOS si la documentation pour le portage de RTEMS est insuffisante.

4.2 Remote debugging

Le fait de concevoir et debugger une application sur un système hôte, en général un PC, qui n'est pas le système cible (système sur lequel l'application est exécutée) complique quelque peu le processus de développement. Il est nécessaire d'installer toute une chaine de logiciels afin que les deux systèmes puissent échanger des informations dans le but de voir sur l'écran du PC ce qui se passe exactement dans le système cible pendant l'exécution de l'application.

Travailler avec un **IDE** (Integrated Developpement Environement) permet d'incorporer tous ces logiciels dans une même interface utilisateur, rendant le processus de conception moins laborieux Il suffit de configurer cet IDE et il se charge ensuite d'utiliser les programmes de la chaine au moment opportun.

Le code c/c++ standard écrit dans l'éditeur de l'IDE doit en premier lieu être traduit en un code machine exécutable par la cible. Cela requiert donc un compilateur et un linker spécifiquement développés pour la cible. Un fichier de commandes spécifique à la cible doit aussi être fourni au linker. Ces deux programmes transforment touts les fichiers .h et .c de l'application en un seul fichier .out contenant l'application exécutable. Il faut ensuite pouvoir télécharger ce code dans la mémoire de la cible, cela est expliqué plus bas.

Pour debugger efficacement une application, il est nécessaire de pouvoir stopper son exécution à des instants définis, de pouvoir exécuter une série d'instructions pas à pas et de pouvoir lire les variables en mémoire ou les registres du processeur, tout cela à partir de l'IDE. L'exemple concret de la lecture de la valeur d'une variable en mémoire sera pris pour expliquer le fonctionnement de la chaine.

L'IDE envoie une commande de type print nom_variable à un debugger spécifique à la cible. Celui-ci, à l'aide du fichier .out va déterminer le type et l'adresse mémoire de la variable et envoyer une commande read memory en protocole RSP (Remote Serial Protocol). Ce paquet RSP est envoyé à un port TCP défini par un nouveau programme, un OCD (On Chip Debugger). Ce programme est un **serveur daemon** qui transforme les requêtes RSP en des commandes JTAG adaptées à une interface matérielle ciblée et les envoie par un port de sortie du PC (USB, série ou parallèle). Cette interface matérielle appelée dongle crée le lien physique entre le PC et la cible et est nécessaire pour transformer les signaux venant du PC en des signaux purs JTAG (JTMS, JTCK,…) adaptés à la cible. Ces signaux vont récupérer de la mémoire de la cible la valeur de la variable grâce au protocole JTAG. La valeur demandée va faire le chemin inverse pour être enfin affichée sur l'écran du PC.

La Figure 24 résume toute cette toolchain.

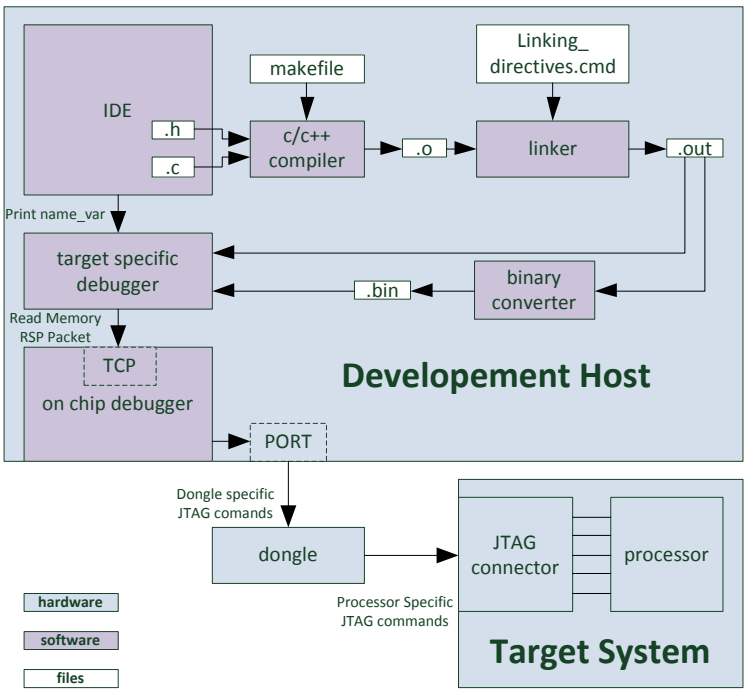


Figure 24 : Toolchain remote debug

Il est à noter que c'est le même procédé qui est utilisé pour télécharger l'application compilée dans la cible. Le fichier .out doit d'abord être converti en un fichier binaire .bin par une application supplémentaire avant d'être écrit en mémoire. Cette application est OBJOCPY.

4.3 Installation de la toolchain

Comme mentionné dans le chapitre 3.2.2, le département d'Infotronique travaille sur un projet nommé NTRT, qui utilise une carte dont le processeur principal est le STM32F103ZET6, le même utilisé sur la carte bi-cortex. Un environnement de développement complet et fonctionnel utilisant Eclipse a donc déjà été mis en place. Il sera donc utilisé et modifié afin de permettre le debuggage de deux processeurs simultanément.

L'IDE utilisé est Eclipse dans sa version Galileo build 20090920-1017. La toolchain Yagarto fournit le compilateur/linker c/c++ GCC V4.2.2 et le debugger GDB V6.8.50 qui sont compatibles avec les cœurs Cortex-M3. Enfin OpenOCD est utilisé comme on chip debugger. L'installation de tous ces logiciels est très clairement décrite dans le fichier d'aide aide_NTRT.chm, écrit par Xavier Meyer et produit pour NTRT. Il est présent dans le répertoire /software/installation_IDE du cd-rom de même que les installeurs nécessaires.

Deux versions d'Eclipse déjà préconfigurées se trouvent dans le cd-rom au répertoire \software\installation_IDE\Installeurs\Eclipse. Il suffit de les copier sur la machine hôte, d'importer les deux

projets compressés (un pour chaque processeur) situés dans \software\installation_IDE\Projets_bicortex et de faire correspondre le répertoire de workspace utilisé à celui utilisé dans la configuration de GDB.

Le répertoire d'OpenOCD fourni sur le cd-rom dans \software\installation_IDE\Installeurs inclut déjà les fichiers Bicortex.cfg et Bicortex2.cfg décrits plus bas. Il suffit de copier ce dossier dans c:\ apps.

L'utilisation de ces programmes pré-configurés dispense de la lecture des 2 chapitres suivants. Cependant, il est préférable de les lire afin d'identifier les changements effectués et leur raison.

Les « logiciels » FTD2xx, Virtual COM STM32, et Realterm ne doivent pas être installés.

4.3.1 Configuration d'OpenOCD pour Eclipse

Il y a deux différences notables entre la carte développée dans ce travail de diplôme et la carte NTRT. En premier lieu, la carte NTRT ne nécessite pas de dongle pour être connectée au pc vu qu'elle intègre directement une puce FTD2232 de l'entreprise FTDI, puce qui se trouve justement dans les dongles habituellement. Le dongle JTAGkey produit par Amontec à base de cette même puce a donc été choisi pour connecter la carte bicortex au PC via le port USB. Celui-ci nécessite l'installation de drivers spécifiques aussi disponibles dans le cd-rom dans \software\installation_IDE\drivers _JTAGKey avec leur notice d'installation. Il faut, en outre, modifier la configuration d'OpenOcd pour qu'il produise des commandes sur le port USB interprétables par le JTAGKey.

La présence de deux processeurs avec deux chaines JTAG distinctes requiert le lancement de deux instances d'Eclipse et d'OpenOCD qui écoutent à des ports TCP différents. Cela pour permettre le debugage simultané des deux applications qui tournent sur chacun des cœurs.

L'implémentation de ces deux différences est décrite ici. Une fois que l'installation d'Eclipse et des plug-ins additionnels est terminée, il faut créer un double du répertoire complet d'Eclipse pour avoir deux versions indépendantes.

Chaque version doit travailler dans un workspace différent.

Avant de configurer les deux instances d'OpenOCD dans Eclipse, il faut effectuer quelques modifications dans le répertoire source d'OpenOCD.

- Allez dans le répertoire C:\Apps\openocd\share\openocd\scripts\board et effectuez une copie du fichier NTRT.cfg.
- Placez deux fois cette copie dans le répertoire C:\Apps\openocd\
- Renommez ces copies en bicortex.cfg et bicortex2.cfg
- Modifiez chacun de ces fichiers de la sorte :
    - Dans les deux fichiers remplacez la ligne :
      source [find share/openocd/scripts/interface/jtagNTRT.cfg]
      par
      source [find share/openocd/scripts/interface/jtagkey.cfg]
      Cela indique à OpenOCD que les commandes qu'il va envoyer passeront par le port USB et devront avoir le format attendu par le JTAGKey.

    - Dans bicortex2.cfg uniquement remplacez la ligne :
      gdb_port 3333
      par
      gdb_port 3332
      Cette modification permettra de lancer une instance d'OpenOCD qui écoutera au port TCP 3332.

La partie Lancement et configuration OpenOCD de l'aide peut maintenant être suivie. Il faut s'assurer d'avoir pour chacune des instances d'Eclipse la configuration suivante (Figure 25).

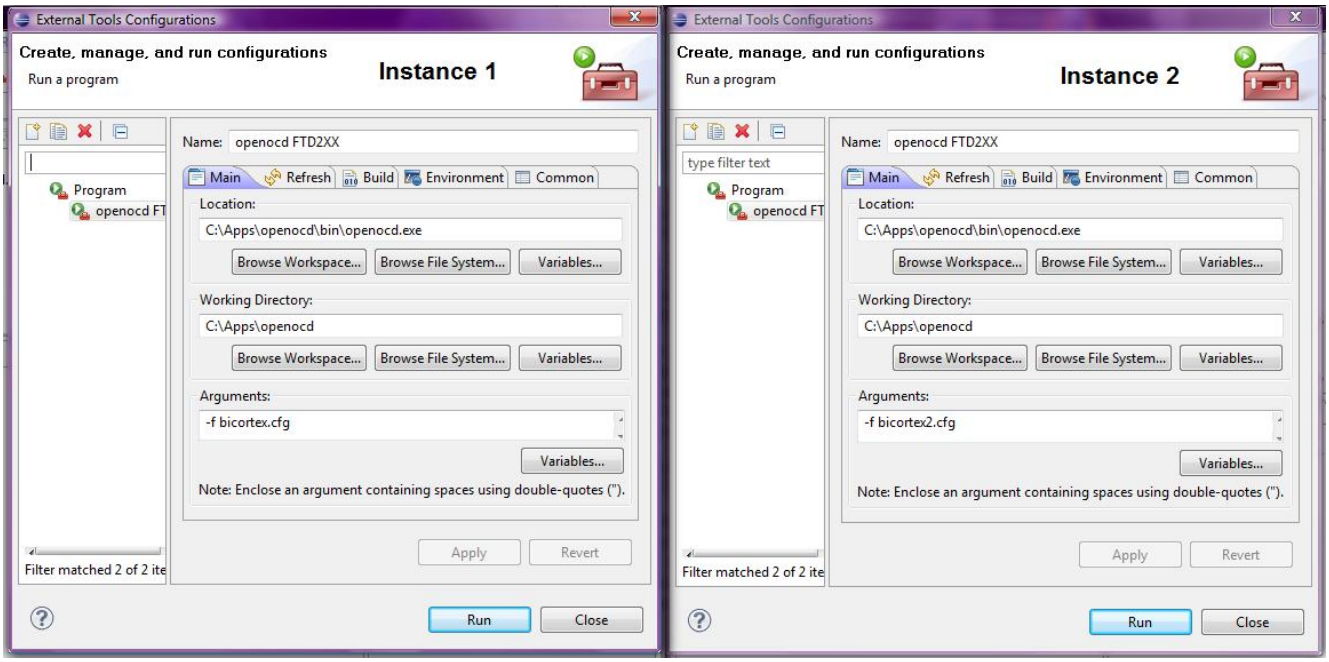


Figure 25 : Configuration OpenOCD

4.3.2 Configuration de GDB pour Eclipse

La partie Lancement et Configuration Debuger explique la configuration de GDB. Cependant, des changements doivent être effectués pour répondre aux spécificités du système.

La commande soft_reset_halt utilisée par GDB pour reseter le processeur a été remplacée par la commande reset halt pour réparer un bug (expliqué au chapitre 4.4.3).

Le port TCP d'envoi de la deuxième instance de GDB doit être changé pour correspondre au port d'écoute configuré pour la seconde instance d'OpenOCD.

Effectuez tous les contrôles indiqués dans le fichier d'aide pour les deux instances d'Eclipse. Avant de cliquer sur APPLY, pour les deux instances, dans l'onglet startup remplacez la ligne *mon soft_reset_halt* par *mon reset halt* (sans underscore). Enfin pour l'instance 2, (celle ou la configuration d'OpenOCD se fait par le fichier bicortex2.cfg), remplacez la ligne *target extended-remote localhost:3333* par *target extended-remote localhost:3332*.

Cliquez sur Apply puis Close.

4.3.3 Procédure de connexion carte-PC

Pour garantir la bonne connexion entre le PC et les deux processeurs, il est nécessaire de suivre un ordre branchement des JTAGkey et de lancement des deux instances d'OpenOCD précis :

- Lancer les deux instances d'Eclipse avec les applications pour chacun des processeurs.
- Débrancher les deux JTAGKey.
- Connecter le premier JTAGKey et lancer OpenOCD depuis une des deux instances d'Eclipse. Cette instance d'Eclipse est maintenant connectée au processeur branché.
- Connecter le second JTAGKey et lancer OpenOCD depuis l'autre instance d'Eclipse. Cette dernière est maintenant connectée avec sur processeur connecté en second.

Il est impératif de débrancher les deux JTAGKey, de fermer toutes les instances d'OpenOCD et de recommencer la procédure (sauf le lancement des deux instances d'Eclipse) à chaque redémarrage d'une instance d'OpenOCD.

4.4 Configuration software du matériel

Maintenant que le code écrit dans Eclipse peut être compilé, téléchargé et exécuté sur un processeur de la carte, il faut configurer les différents périphériques du processeur pour qu'ils utilisent le matériel présent sur la carte. Cette configuration se fait par la modification des différents registres de contrôle dédiés à chaque périphérique. Mais plutôt que d'accéder directement à ces registres, STm a écrit des fonctions qui effectuent les changements plus intuitivement.

Toutes les fonctions et les variables utilisées lors de cette configuration sont données par la librairie spécifique au processeur téléchargeable sur le site de STm et présente sur le CD-ROM dans le répertoire /software/STM32f10x_stdperiph_Lib. Elles sont toutes précisément décrites dans le fichier d'aide (stm32f10x_stdperiph_lib_um.chm) inclus dans ce répertoire (sous l'onglet Modules dans la partie respective à chaque périphérique).

La configuration d'un périphérique se fait en 5 étapes distinctes :

1.) Enclenchement du clock spécifique au périphérique
2.) Déclaration des structures servant à configurer le périphérique
3.) Attribution des valeurs aux membres de ces structures
4.) Chargement de ces valeurs dans les registres concernés
5.) Enclenchement du périphérique configuré

L'arbre d'horloge n'étant pas vraiment un périphérique sa configuration est un peu différente.

Le code configurant ces périphériques est suffisamment commenté pour ne pas nécessiter d'explication trop détaillée.

Par manque de temps, le port USB n'a pas été utilisé et donc il n'a pas non plus été configuré.

4.4.1 Configuration de l'arbre d'horloges (RCC)

Par défaut, le processeur utilise des oscillateurs internes pour générer son horloge principale et RTC (Real Time Clock). Il faut donc définir l'oscillateur externe à 8Mhz (HSE) comme source d'horloge principale et le quartz à 32,768 khz (LSE) comme source pour la RTC.

De plus, différents multiplicateurs et multiplexeurs doivent être configurés pour déterminer l'horloge du système (SYSCLK), des **bus AHB**, **APB1** et **APB2,** ainsi que l'horloge du contrôleur USB (USBCLK) qui doit toujours valoir 48Mhz. L'arbre complet des sources d'horloge du processeur est disponible en annexe XXII.

Il est important de constater que la configuration des accès à la mémoire flash est dépendante de SYSCLK (STM32F10xx_advanced_datasheet.pdf pages 48-49), cette configuration est donc réalisée ici.

La méthode void RCC_RTC_initial(void) définie dans les fichiers RTC_RCC_Config.h et .c se charge d'effectuer ces changements. Son Implémentation est consultable en annexe XXIII.

4.4.2 Configuration des pins (GPIO)

Le code la configuration du périphérique PORTC utilisé pour driver les leds visible ici peut servir de référence sur la marche à suivre pour configurer correctement un périphérique pour les processeurs STM32F10x. On y retrouve les 5 étapes décrites plus haut. L'activation du périphérique n'est pas nécessaire pour les ports d'I/O.

```
void LED_initial(void)
{
    //I. ENABLE USED PERIPHERAL CLOCK
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);

    //II. DEFINE INIT STRUCTURES FOR USED PERIPHERALS
    GPIO_InitTypeDef GPIOC_InitStructure;

    //III. SET INIT STRUCTURES FOR USED PERIPHERALS
    GPIOC_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; //output push/pull
    GPIOC_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//max speed allowed

    //IV. INITIALIZE PERIPHERAL FOR EACH PIN
    GPIOC_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_Init(GPIOC, &GPIOC_InitStructure);
    GPIOC_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_Init(GPIOC, &GPIOC_InitStructure);
    GPIOC_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_Init(GPIOC, &GPIOC_InitStructure);
    GPIOC_InitStructure.GPIO_Pin = GPIO_Pin_3;
    GPIO_Init(GPIOC, &GPIOC_InitStructure);

    //V. ENABLE PERIPHERAL not needed
}
```

Figure 26 : Exemple de configuration d'un périphérique

Chaque pin de ce processeur peut être configuré indépendamment, non seulement pour sa direction (input/output) mais aussi pour ses caractéristiques électriques (pull-up/down, floating, analog,…). Certains pins, comme ceux utilisés par le contrôleur mémoire ont une utilisation alternative qui doit être activée. Il faut aussi spécifier la fréquence maximale pouvant arriver sur ces pins, la valeur maximale de 50Mhz a été choisie.

Chaque pin doit être configuré l'un après l'autre.

Les fichiers GPIO_config.h et .c implémentent trois méthodes différentes pour configurer les trois séries de pins utilisées par les boutons, les leds et le contrôleur de mémoires externes.

La méthode void BUT_initial(void) configure les pins du port A utilisés par les boutons. Ces pins doivent être en direction INPUT et la pull-up doit être activée.

La méthode void LED_initial(void) configure les pins du port C utilisés par les leds. Ces pins doivent être en direction OUTPUT et en mode push/pull.

La méthode void FSMC _IO_initial(void) configure des pins des ports D, E et F utilisés par le FSMC. Tous ces pins sont configurés en mode alternative push-pull sauf le pin NWAIT qui doit être en mode input pull-up comme cela est visible au tableau 29 du datasheet STM32F10xx_advanced_datasheet.pdf.

Le code implémentant ces trois fonctions est visible en annexe XXIV.

4.4.3 Configuration du contrôleur mémoire (FSMC)

Le contrôleur de mémoire étant très polyvalent, il faut lui indiquer toutes les caractéristiques de la mémoire qu'il pilote. Largeur de bus de donnés, mode et temps d'accès ou utilisation du signal d'attente (NWAIT) sont autant de paramètres qu'il faut configurer ainsi que bien d'autres.

Il faut en premier lieu déterminer quel mode d'accès proposé par le contrôleur permet d'utiliser correctement la mémoire choisie. Ces modes sont décrits par des chronogrammes aux pages 420 à 438 du datasheet STM32F10xx_advanced_datasheet.pdf. Tous les modes synchrones (utilisant un signal d'horloge) peuvent être exclus immédiatement. Les modes 2, B, C, D et multiplexé utilisent un signal (NADV) qui n'est pas disponible sur la mémoire choisie, ils sont donc aussi à exclure.

Il reste donc les modes 1 et A. Tous deux sont compatibles avec les spécifications de la mémoire. Seule la séquence d'écriture de ces deux modes diffère. En mode1, le signal NOE est placé immédiatement à 0, appliquant au bus de données des valeurs pas encore stabilisées. Le mode A au contraire attend un temps configurable avant de placer les données sur le bus. Ce dernier mode a donc été choisi pour sa plus grande modularité. Le signal NBL[1 :0] n'est utilisé que pour une mémoire dont la largeur de bus de données est de 16 bits, ce qui n'est pas le cas avec la mémoire choisie. Il sert déterminer le byte de poids fort ou faible d'une donnée de 16 bits pour effectuer des accès par byte.

La superposition des timings de la mémoire avec les chronogrammes décrivant le Mode A est visible en Figure 27 et Figure 28.

ADDSET (AST) et DATAST (DST) sont les deux seuls paramètres de timing configurables. Il faut leur attribuer les valeurs minimales qui respectent toutefois les spécifications données dans le datasheet de la mémoire.

En sachant que le processeur travaille à une fréquence de 72 Mhz, HCLK = 1/72Mhz = 13.88ns peut être déterminé. Pour un accès en lecture (read acces), il ressort que cette mémoire est suffisamment rapide pour ne nécessiter aucun temps d'attente supplémentaire. ADDSET et DATAST pourraient théoriquement être configurés à 0. Le temps minimal de 1 HCLK pour l'établissement de l'adresse et des données est suffisant à garantir la stabilité des données. Les flèches bleues montrent dans cette configuration les marges de temps obtenues dans ces conditions.

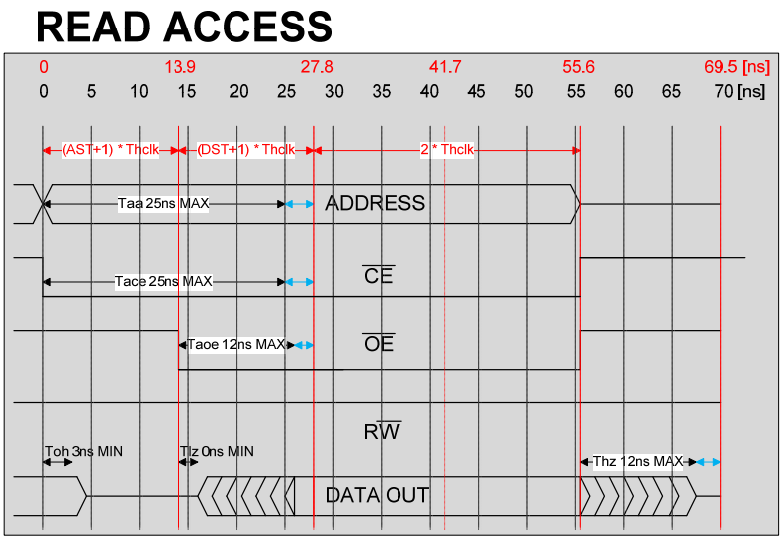


Figure 27 : Read timings

Pour l'accès en écriture (write acces), la durée minimale à respecter pour Twp de 20ns impose la valeur minimale de 1 pour DATAST. La valeur de 0 pour ADDSET convient toujours. Les flèches bleues sur la Figure 28 montrent las marges avec ces valeurs.

Il faut donc attribuer à ADDSET la valeur 0 et à DATAST la valeur 1 pour respecter toutes les conditions d'accès. Cela se fait en attribuant les bonnes valeurs aux paramètres FSMC_AddressSetupTime, et FSMC_DataSetupTime.

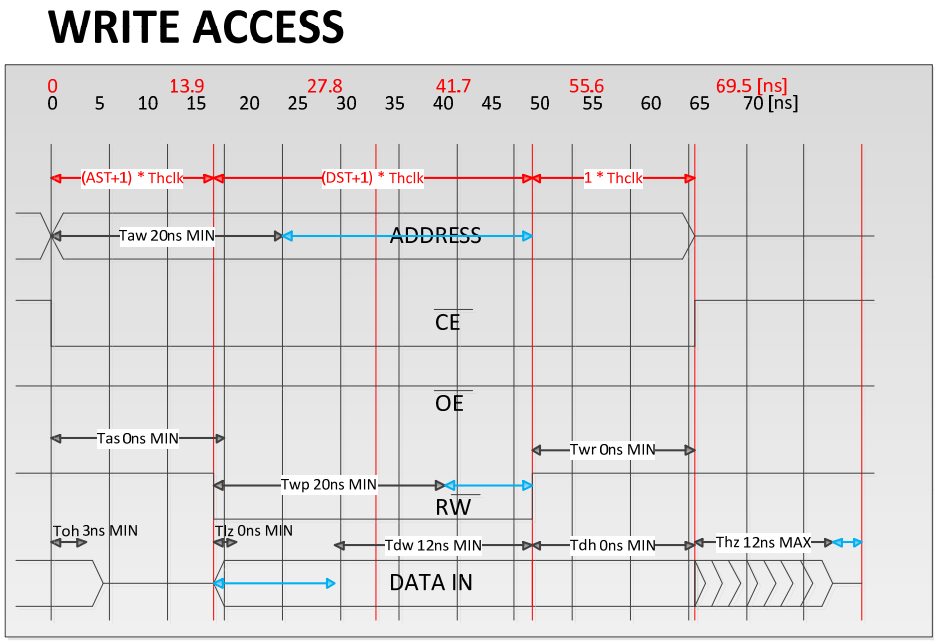


Figure 28 : Write timings

Un autre paramètre particulier mérite une explication : la variable FSMC_WaitSignalActive spécifie si le premier waitstate dû au changement d'état du signal NWAIT doit être introduit un cycle après ce changement d'état ou immédiatement. Aucune documentation n'as permis de savoir quel paramètre choisir, il a donc été laissé à sa valeur par défaut. La mémoire fonctionne correctement, l'autre possibilité n'a donc pas été testée.

Un problème est survenu lors de la configuration des timings de la mémoire. Un premier programme configurant les paramètres FSMC_AddressHoldTime, FSMC_AddressSetupTime, FSMC_DataSetupTime à leur valeur maximale (0xF ou 0xFF) pour être sur de respecter les timings de la mémoire a été testé, la mémoire fonctionnait parfaitement mais plus lentement. Ce programme a ensuite été modifié afin d'optimiser les temps d'accès en mettant à zéro les deux premiers paramètres et à 1 le dernier. Le fichier d'aide de la librairie standard indique que tous ces paramètres peuvent être mis à zéro.

Le programme a planté et le debuger n'arrivait plus à reseter le processeur pour reprogrammer la mémoire. Après avoir essayé quelques solution peu orthodoxes (couper l'alimentation de la mémoire, puis les pins CE et OE) sans succès, l'option de produire une nouvelle carte a été envisagée. Marc Pignat, spécialiste JTAG du département Infotronique, a alors essayé de changer la commande soft_reset_halt envoyée par le debuger par la commande reset halt. Cela a fonctionné!

La commande soft_reset_halt n'est en fait pas standardisée et buguait dans ce cas précis. Il a été constaté par la suite (dans le datasheet STM32F10xx_advanced_datasheet.pdf aux pages 442 et 444) que la variables FSMC_AddressHoldTime ne peut pas valoir zéro. Le fichier d'aide contient donc des erreurs…

Les fichiers FSMC_config.h et .c proposent une méthode void FSMC_initial(void) qui réalise cette configuration.

Le code de ces fichiers est en annexe XXV.

4.4.4 Configuration des interruptions externes (EXTI)

Il faut ici activer les lignes d'interruptions nécessaires et configurer le type de signal interrompant (flanc/niveau) et son niveau (haut/bas) ou flanc (montant/descendant). Il faut aussi spécifier s'il s'agit d'un évènement ou d'une interruption. Une interruption stoppera automatiquement l'exécution de l'application courante tandis qu'un événement doit être **pollé** pour être traité.

Les boutons produisent un signal actif à l'état bas et l'interruption est configurée pour être calée sur le flanc descendant du signal. Les lignes EXTI_Line0, EXTI_Line1, EXTI_Line2 et EXTI_Line3 activent les interruptions sur les pins correspondants du port spécifié dans le registre AFIO_EXTICR1 (STM32F10xx_advanced_datasheet.pdf page 167). Par défaut, c'est le PORT A qui est utilisé. Ce registre n'est donc pas modifié.

Les fichiers EXTI_config.h et .c définissent une méthode void EXTI_initial(void) qui configure ce périphérique. La ligne d'interruption attribuée à la mémoire n'est pas activée par cette méthode.

L'implémentation de cette méthode est consultable en annexe XXVI.

4.4.5 Configuration du contrôleur d'interruptions (NVIC)

Par défaut, évidement, aucune ligne d'interruption configurable n'est activée, seules les interruptions non-masquables sont détectées. Il faut donc activer les lignes d'interruptions qui arrivent de l'EXTI configuré plus haut. Il faut aussi définir quel est le format de définition (nbre de bits) de priorité/préemption qui sera utilisé et quelle sera la priorité des interruptions activées. De plus, il est impératif de spécifier où se trouve la table des vecteurs d'interruption.

Le modèle d'interruption utilisé dans le Cortex-M3 est celui de l'interruption vectorisée. C'est-à-dire que lors d'une interruption, le NVIC va automatiquement placer dans le registre Program Counter (PC), l'adresse qu'il trouvera dans la table des vecteurs d'interruption à l'endroit désigné pour l'interruption survenue. Cette adresse pointe vers la routine d'interruption spécifique qui gère cette interruption. La table des vecteurs est donc un élément central du mécanisme d'interruption. Elle se trouve par défaut dans la flash à partir de l'adresse 0x0000 0000. Cela est défini dans le fichier de linking_directives.cmd configurant le linker. C'est la raison pour laquelle lors d'une pression sur le bouton reset, le PC est remis à 0x0000 0000, le processeur exécute l'instruction fetch top stack value puis saute au Reset Handler dont l'adresse se trouve à 0x0000 0004.

La préemption des interruptions n'étant pas utilisée le format de priorité 0 (4 bit de priorité, 0 pour la préemption) est utilisé. La priorité de préemption est donc de 0 et la priorité des interruptions des boutons est fixée à 10 arbitrairement juste pour être moins prioritaire que les interruptions non-masquables.

Les fichiers NVIC_config.h et .c proposent une méthode void NVIC_initial(void) qui configure de la sorte le contrôleur d'interruptions.

Ces fichiers sont joints en annexe XXVII.

4.5 Application basique de test des fonctionnalités

Une petite application pour vérifier que ces différents périphériques fonctionnement bien a été développée : appuyer sur un bouton géré par le processeur de gauche fait allumer la led correspondante gérée par le second processeur. Son implémentation est basique. En fonction du bouton appuyé, le premier processeur écrit une valeur dans un emplacement de la mémoire dual port, le second processeur effectue un polling du contenu de cet emplacement et allume la led correspondante en fonction de la valeur lue. Cette application fonctionne exactement comme prévu, cela montre que:

- Les pins d'I/O sont configurés correctement car tous les périphériques externes fonctionnent.
- La mémoire est configurée correctement, les deux processeurs y ont accès et le signal *busy* produit bien son effet.
- Les interruptions générées par les boutons sont bien prises en compte autant par le EXTI que par le NVIC et traitées par leur handler respectifs.

Le code de cette application (fichiers test_app.c et app_int.h et .c) se trouvent en annexe XXVIII.

Le fichier test_app.c contient l'application principale avec sa machine d'états. Les deux autres contiennent les routines d'interruption qui gèrent l'appui sur les boutons.

4.6 Application de démonstration

Avoir à disposition une grande puissance de calcul est une chose, l'exploiter au maximum en est une autre. Jusqu'à l'apparition des OS, les systèmes informatiques ne pouvaient exécuter qu'une application à la fois et cette dernière était conçue comme une suite séquentielle d'opérations aboutissant à un résultat. Chaque opération dépendant de l'effet de l'opération précédente, il est impossible de faire exécuter un code de ce type de façon parallèle sur deux processeurs.

Les OS modernes ont apporté ce qui est communément appelé le multi-tâches. Ce terme ne correspond toutefois pas à la réalité si le système ne dispose que d'un cœur. Un cœur n'est jamais capable d'exécuter quoi que ce soit en parallèle. En réalité, les OS placent chaque application dans un thread (ou plusieurs), puis un logiciel (intégré à l'OS) appelé scheduler s'occupe d'attribuer le seul cœur du système à chacun un des threads alternativement pendant un temps défini (par exemple 1ms). Les applications s'exécutent donc toutes petit à petit mais séquentiellement et pas de façon vraiment parallèle. Ce n'est que le temps très court attribué à chaque tâche qui donne l'impression de simultanéité.

La Figure 29 illustre ce principe.

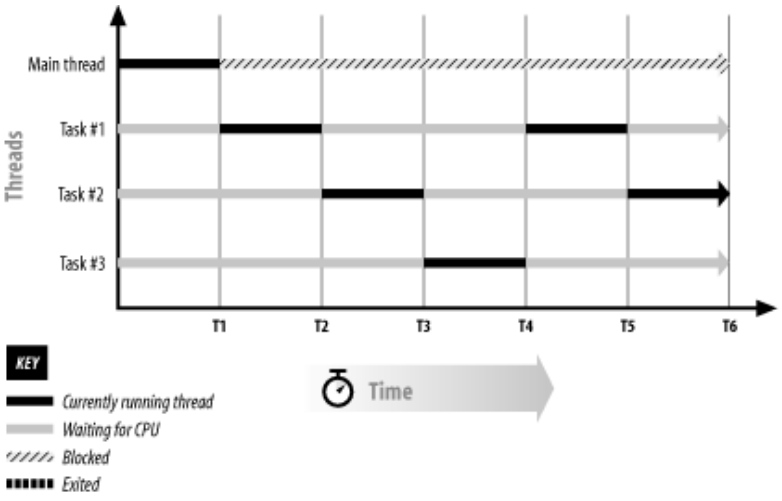


Figure 29 : Illustration scheduler

Sur un système disposant de 2 ou plusieurs cœurs, un véritable parallélisme peut être réalisé, chacun des cœurs pouvant travailler indépendamment de l'autre. Mais pour qu'ils travaillent afin arriver à un résultat unique et commun, à un moment ou à un autre il est obligatoire que les cœurs échangent des informations (attribution du rôle de chacun, valeur de variables initiales ou mise en commun du résultat partiel obtenu). Ces échanges sont appelés overhead de parallélisation. En effet, si le code s'exécutait sur un seul cœur ces échanges n'auraient pas lieu d'être. La parallélisation entraine donc inévitablement une augmentation du code à exécuter.

De plus, échanger les threads qui s'exécutent dans le cœur à un coût supplémentaire du au switching context. Pour qu'un thread puisse recommencer son travail là où il l'avait laissé, tout le contexte dont il se servait doit être restauré (registres, flags de l'ALU, ou variables dans la ram). La restauration sous-entend évidement une sauvegarde préalable et ces deux actions indispensables monopolisent le processeur pour faire autre chose que les tâches réelles qui lui sont assignées.

Pour concevoir un logiciel multi-threadé efficace, il faut donc optimiser les switching context, l'overhead de parallélisation et en général toute communication entre les cœurs pour qu'ils travaillent au maximum sur le code effectif de l'application. Il est bien clair que la notion de switching contex n'est valable que si un OS gère le système.

Une application qui se prêt particulièrement bien à une parallélisation efficace est le calcul des décimales de π avec l'algorithme de Bailey – Borwein - Plouffe, expliquée dans le chapitre suivant.

4.6.1 Calcul des décimales de π en hexadécimal

Contrairement aux algorithmes traditionnels qui utilisent des séries convergentes pour calculer de plus en plus précisément la valeur de π, l'algorithme de Bailey – Borwein – Plouffe permet de calculer chacune des décimales de π en hexadécimal sans devoir calculer aucune des précédentes. C'est pour le moment, avec ses variantes binaires, la méthode la plus efficace de calculer π avec une précision donnée. La page Wikipédia suivante explique en détail la méthodologie pour arriver au résultat http://en.wikipedia.org/wiki/Bailey-Borwein-Plouffe_formula.

Cet algorithme est basé sur l'équation suivante :

$$\pi = 4\sum_{k=0}^{\infty}\frac{1}{(16^k)(8k+1)} - 2\sum_{k=0}^{\infty}\frac{1}{(16^k)(8k+4)} - \sum_{k=0}^{\infty}\frac{1}{(16^k)(8k+5)} - \sum_{k=0}^{\infty}\frac{1}{(16^k)(8k+6)}.$$

Chacun des 4 termes de cette somme peut être décomposé en deux termes, un premier fini et un second infini, en fonction de la décimale n que l'on veut calculer. Par exemple la décomposition du premier terme donne :

$$\sum_{k=0}^{\infty}\frac{1}{(16^k)(8k+1)} = \sum_{k=0}^{n}\frac{1}{(16^k)(8k+1)} + \sum_{k=n+1}^{\infty}\frac{1}{(16^k)(8k+1)}.$$

Il est maintenant possible de multiplier numérateur et dénominateur de chacune de ces sommes par 16^n :

$$\sum_{k=0}^{\infty}\frac{16^{n-k}}{8k+1} = \sum_{k=0}^{n}\frac{16^{n-k}}{8k+1} + \sum_{k=n+1}^{\infty}\frac{16^{n-k}}{8k+1}.$$

Des explications mathématiques montrent que seule la partie fractionnaire de chacune de ces sommes permet de calculer ce résultat. La somme infinie converge très rapidement et est toujours fractionnaire (<1). Pour avoir la partie fractionnaire de la somme finie l'ajout d'une fonction modulo convient parfaitement :

$$\sum_{k=0}^{\infty} \frac{16^{n-k}}{8k+1} = \sum_{k=0}^{n} \frac{16^{n-k} \mod 8k+1}{8k+1} + \sum_{k=n+1}^{\infty} \frac{16^{n-k}}{8k+1}.$$

Un algorithme itératif appelé exponentiation modulaire permet de calculer très rapidement $16^{n-k} \mod 8k + 1$. C'est là que réside le secret de la rapidité de cette méthode de calcul de π.

Il faut donc calculer les 4 termes de la première équation avec cette méthode puis effectuer la somme totale :

$$4\Sigma_1 - 2\Sigma_2 - \Sigma_3 - \Sigma_4.$$

Il suffit à nouveau de ne garder que la partie fractionnaire de cette somme, de la multiplier par 16 et la partie entière cette fois du résultat donne en hexadécimal la valeur de la décimale cherchée n.

Voici les 16 premières décimales de π en hexadécimal 3.243F 6A88 85A3 08D3. Pour obtenir la valeur décimale de π il faut convertir normalement d'hexadécimal en décimal. Par exemple la conversion de 3.243F donne :

3*16^{0} + **2***16^{-1} + **4***16^{-2} +**3***16^{-3} + **15***16^{-4} = 3,141586 ≈ 3,1415926…

4.6.2 Programme développé

L'application développée pour la carte de démonstration est une variante de cet algorithme optimisée pour ne pas utiliser de variables à virgule flottante et qui permet de calculer plusieurs décimales en une passe en fonction de la précision permise par les types de variables choisis. Elle est en outre optimisée pour un système à deux processeurs. L'utilisateur a le choix, avant de lancer le calcul, d'utiliser un seul ou les deux cœurs.

Avec les paramètres de précision configuré pas défaut, 8 décimales en hexadécimal sont calculées à chaque passe. Cependant seules les 4 de poids fort sont exactes. Des problèmes d'arrondi lors des divisions entières et l'utilisation de résultats tronqués à 32 bits sont la cause de cette imprécision. Il est d'ailleurs possible d'obtenir autant de décimales correctes en effectuant moins de calculs, particulièrement en ne calculant pas de termes des parties infinies. Le but n'est pas ici d'optimiser le code au maximum, mais de montrer le gain apporté par le second processeur. Dans le reste de l'explication, lorsqu'il est fait mention d'indice d'une décimale c'est de 4 décimales qu'il faut l'entendre et pour cette raison, les indices sautent de quatre en quatre.

Si un seul cœur est utilisé, un nombre défini de décimales est calculé et stocké dans un tableau en ram. Aucun échange entre les deux processeurs n'a lieu. Seules les leds du processeur 1 montrent son activité en clignotant chacune jusqu'à ce que chaque quart des décimales soit calculé.

Si les deux cœurs sont utilisés, deux phases viennent s'ajouter au calcul effectif. D'abord le premier processeur communique au second le nombre de décimales qu'il devra calculer. Le processeur 1 va ensuite écrire dans la mémoire externe tous les indices des décimales que le second processeur va devoir calculer. Il est à noter que plus cet indice est élévé, plus le calcul pour cette décimale sera long. C'est pourquoi il a été décidé que les processeurs calculeront les décimales par alternance. L'un calculera les décimales d'indices 1,9, 17, … l'autre celles d'indices 5, 13, 21,… Cette répartition est la plus équitable.

Lorsque les deux processeurs ont terminé leurs calculs, une phase de rassemblement des résultats a lieu. Le premier processeur communique au second le nombre de ses dernières décimales calculée qu'il doit placer en mémoire partagée. Le premier processeur les récupère alors et au final toutes les décimales voulues se trouvent dans le tableau en mémoire du processeur 1.

4.6.3 Détails d'implémentation

Le fichier app1.c est le code exécuté par le processeur 1. Visible en annexe XXIX, il met à disposition plusieurs fonctions qui effectuent chacune une partie du travail. Au début de ce fichier, 4 paramètres permettent de configurer le nombre de décimales calculées lors de chaque passe.

La valeur définie dans *nbrHexToCompute* qui fixe le nombre de décimales à calculer en une passe impose les valeurs dans les paramètres suivants pour obtenir des résultats cohérents. Ainsi *OneShiftMax* doit être de $16^{\wedge nbrHexToCompute}$ et sert à calculer les termes de la partie infinie de la somme. *Shift* doit valoir *4* nbrHexToCompute,* c'est le nombre de bits de shift qu'il faut appliquer au numérateur de la somme finie pour obtenir un résultat valide lors de la division par le dénominateur. Enfin mask doit valoir *OneShiftMax -1*, il sert à tronquer chaque élément de la somme finie au nombre de bits correspondant à *nbrHexToCompute.*

Les fonctions implémentées dans cette application et simplifiant l'algorithme principal sont les suivantes :

void **extract_digit**(*uint32_t* dec,*uint8_t** table) sert à stocker dans un tableau temporaire passé en paramètre (*table)* les 4 valeurs hexadécimales exactes contenues dans le paramètre *dec*. Cette fonction extrait toujours 4 valeurs hexadécimales. Cela est nécessaire car, certaines décimales de π valent 0 et cette méthode doit pouvoir les extraire dans tous les cas.

uint32_t **calc_16PowxMody**(*uint32_t* power, *uint32_t* modulo) calcule efficacement $(16^{\wedge x})\%y$. Le paramètre power transmet la puissance de 16, tandis que le paramètre modulo indique l'opérande du modulo. La valeur retournée est le résultat de l'opération.

uint32_t **calc_16Powx**(*uint32_t* x) calcule simplement 16^x et retourne le résultat.

uint32_t **sum**(*uint32_t* sumIdent, *uint32_t n*) est la fonction qui calcule les quatre sommes partielles vues plus haut. Le paramètre sumIdent peut prendre les valeurs 1, 4, 5 ou 6 en fonction de la somme partielle calculée. Le paramètre *n* définit pour quelle décimale la somme doit être effectuée. Cette fonction est partagée en deux parties. La première calcule la partie finie de la somme partielle. En shiftant de 32 bits à gauche le numérateur obtenu, la division avec le dénominateur est rendue entière. Puis un masque est appliqué afin de ne garder que le nombre de bits souhaité dans les paramètres de précision. La seconde partie calcule les termes de la somme infinie jusqu'à ce que la précision des variables utilisées permette une variation de du total. La valeur retournée à la sortie des deux boucles est la somme partielle en décimal.

uint32_t **calc_PiDecN***(uint32_t* n) calcule la somme complète pour une décimale donnée en paramètre *n* en appelant 4 fois la méthode **sum** précédente avec les paramètre nécessaires. La valeur retournée est la somme totale en décimal.

La fonction principale pour le processeur 1, *int* **main** (*void*) dont le structogramme est visible en Figure 30 commence par définir quelques variables, puis initialise les périphériques de la carte. L'utilisateur doit ensuite décider avec les boutons du processeur 1 si un ou deux processeurs seront utilisés pour le calcul. Enfin, la machine d'états correspondant au choix de l'utilisateur sera exécutée et les décimales calculées.

Dans le cas ou deux processeurs sont utilisée, le processeur 1 donne l'ordre au processeur 2 de calculer un nombre de décimales en écrivant celui-ci à un endroit défini de la mémoire externe. Ensuite il écrit dans la mémoire externe la liste des index des décimales dont il délègue le calcul. Seulement alors le processeur 1 commence effectivement à calculer les décimales qu'il s'est attribuées. Lorsqu'il a fini, il demande au processeur 2 de lui transmettre un certain nombre des dernières décimales qu'il a calculées afin de rassembler les résultats. Le code de l'application du processeur 2 app2.c est en annexe XXX.

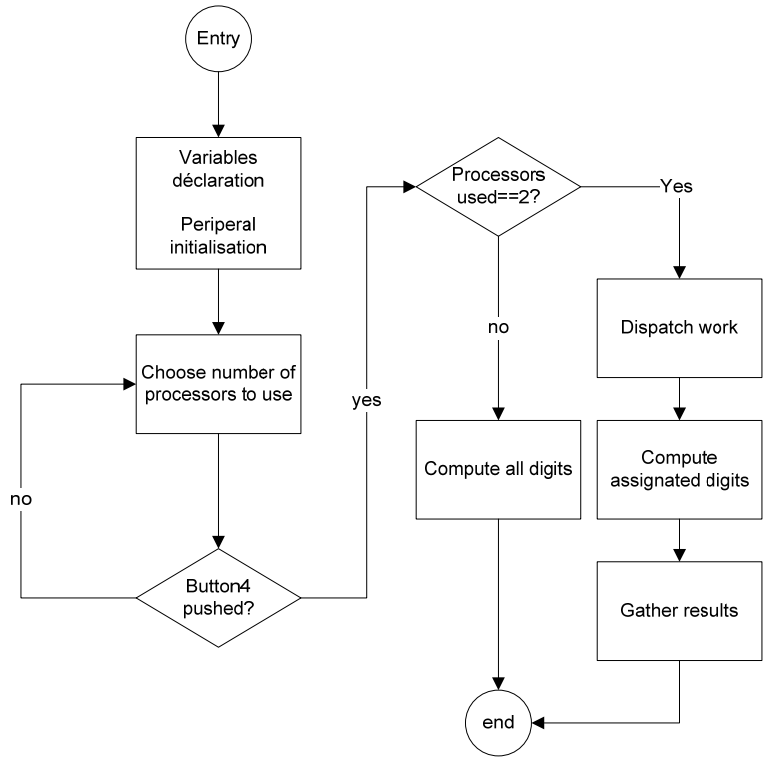


Figure 30 : Structogramme main processeur1

La fonction principale pour le processeur 2 est différente de la précédente, ce processeur attendant des ordres et des données pour s'exécuter. Le stroctogramme de cette fonction principale est consultable en Figure 31. Après la phase d'initialisation, le processeur 2 attend que le processeur 1 lui donne les données dont il a besoin. Une fois le calcul terminé, il attend à nouveau l'ordre du processeur 1 de lui transmettre ses résultats.

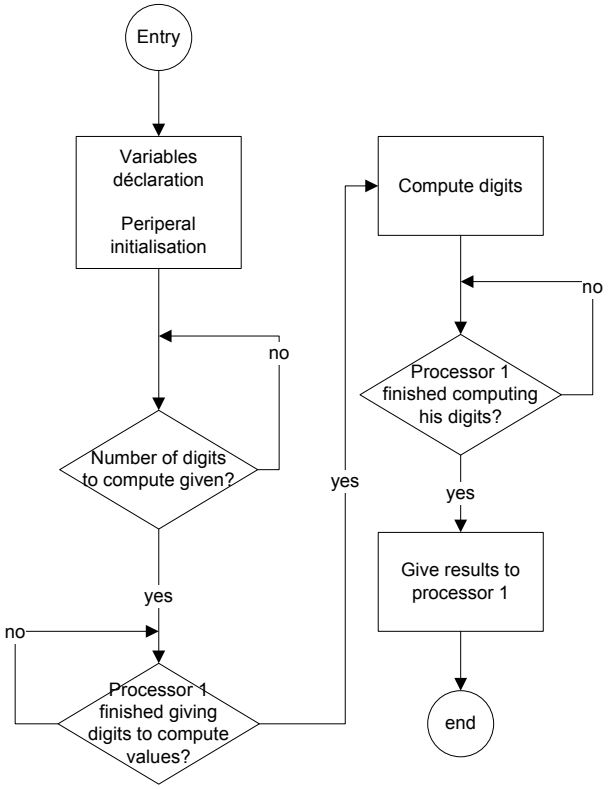


Figure 31 : Structogramme main processeur2

4.6.4 Tests de l'application

Le bon fonctionnement de cette application a d'abord et testé bloc par bloc, c'est-à-dire que les résultats des différentes fonctions utilisées ont étés validés en donnant des paramètres connus en entrées de ces dernières et en comparant la valeur retournée avec la valeur de retour attendue. Toutes les fonctions de cette version de l'application sont implémentées correctement

Ensuite le fonctionnement global de l'algorithme de calcul a été testé en calculant les 500 premières décimales et en comparant le résultat (une trentaine de décimales choisies aléatoirement parmi les 500) avec une liste certifiée disponible à l'adresse suivante :

http://web.student.chalmers.se/~frejohl/pie/pi_b16_100000.txt

La valeur de la 65535 décimale a aussi été calculée pour vérifier que la précision souhaitée était constante sur toute la gamme de valeur. L'algorithme a réussit de ce test (plus de 40 minutes de calcul pour cette seule décimale) comme le montre ce snapshot (Figure 32) de l'arrivée au breakpoint juste après le calcul des 4 décimales 3F60.

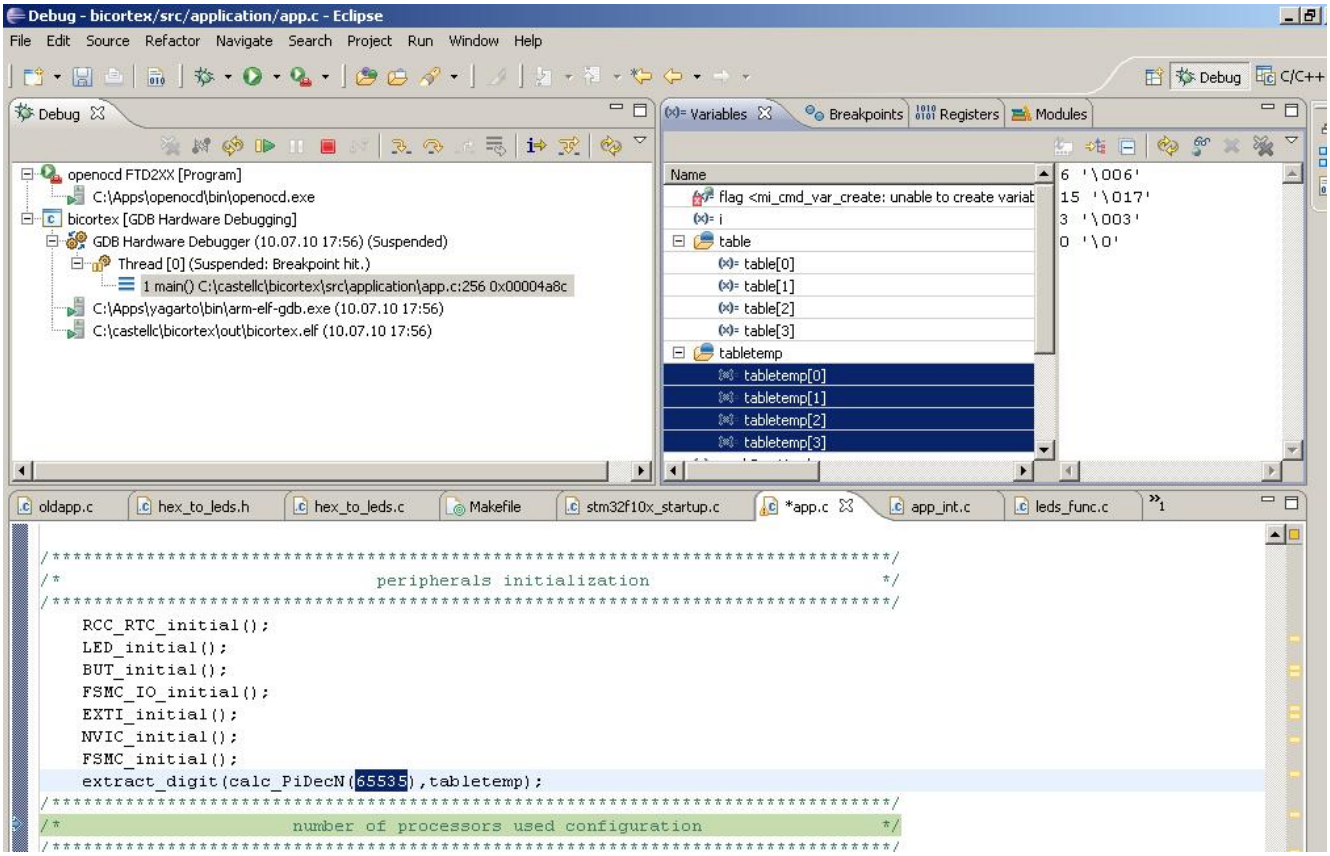


Figure 32 : 4 décimales à partir de 65535

Enfin la collaboration entre les deux processeurs a été testée en faisant calculer la moitié des 500 décimales à chacun des processeurs puis en rassemblant les données dans le tableau du processeur 1 et en comparant à nouveau le résultat avec la liste certifié. Ce test aussi est réussi.

Il est donc possible de certifier que cette application peut calculer toutes les décimales de π en hexadécimal jusqu'à la 65535ème et que la collaboration entre les processeurs est possible et efficace jusqu'au calcul de 508 décimales au total.

4.7 Portage OS temps réel

RTEMS est un OS temps réel très performant, c'est d'ailleurs un des rares OS à être allé dans l'espace. Il a été porté pour un grand nombre de processeurs dont une bonne partie de la famille ARM. Cependant, il ne supporte pas encore l'architecture Cortex-M3. Le travail consiste donc à réécrire tout le code qui a trait au hardware soit au niveau du processeur, soit au niveau des périphériques présents sur la carte. La Figure 33 montre la structure de l'OS et tous les blocs fonctionnels qui accèdent directement au hardware. Ces 5 blocs peuvent se classer dans trois catégories :

- Dépendant du cœur Cortex M3 en rouge
- Dépendant du modèle de processeur STM32F103ZET6 en bleu
- Dépendant de la carte bi-cortex en vert

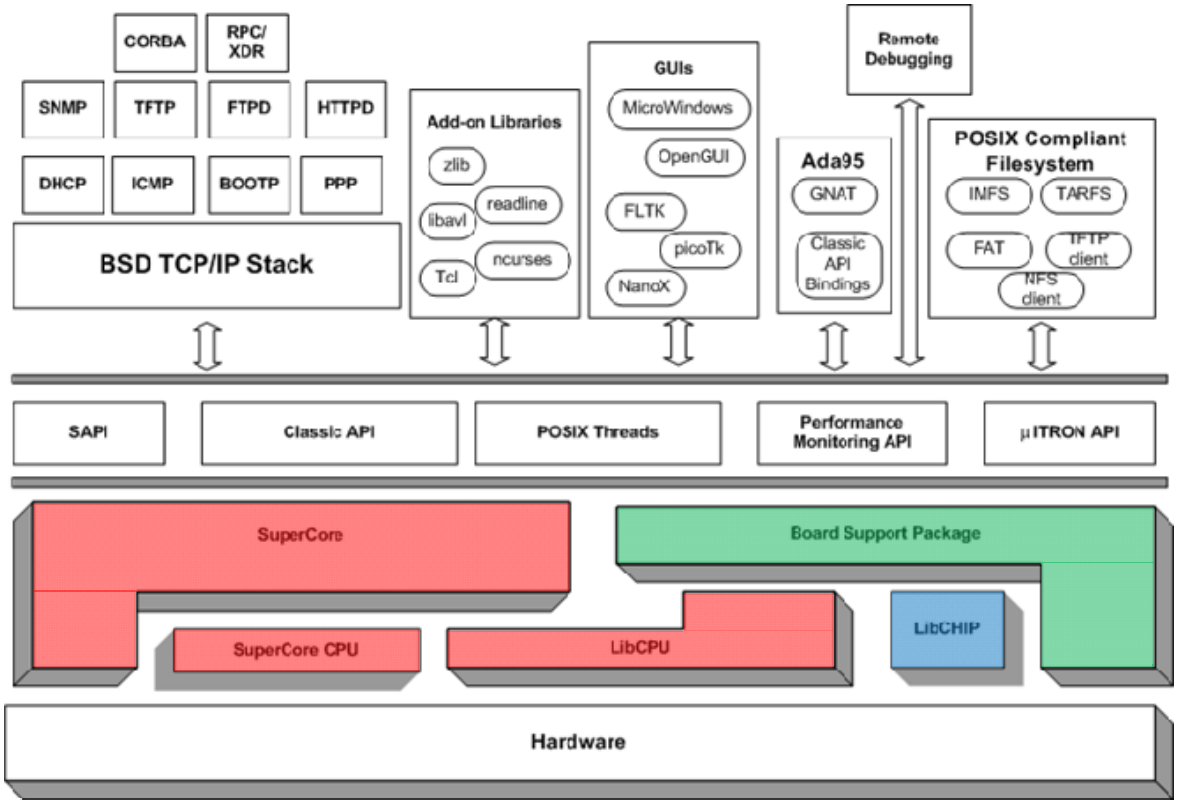


Figure 33 : Architecture logicielle RTEMS

Ce partage en blocs se retrouve dans l'arborescence (relativement complexe) du code source de RTEMS :

- cpukit/score/cpu : fichiers dépendant du cœur du processeur
- c/src/lib/libcpu : fichiers dépendants du modèle du processeur
- c/src/lib/libbsp : fichiers dépendants de la carte mère

Le code source de Rtems 4.9.4 se trouve sur le cd-rom dans \software\rtems-4.9.4

La documentation disponible pour la dernière version complète de RTEMS (4.9.4) se trouve à l'addresse : http://www.rtems.org/onlinedocs/releases/rtemsdocs-4.9.4/share/rtems/html/

Elle est toutefois souvent incomplète, des XXXXXXX jalonnent le texte fréquemment là ou il y aurait du y avoir un mot et les trois formats disponibles différents pour chaque fichier ne contiennent pas les mêmes informations.

Les documents les plus utiles sont :

- Getting Started with RTEMS
- RTEMS BSP and Device Driver Development Guide
- RTEMS CPU supplement (premiers chapitres)
- RTEMS Developement Environement Guide
- RTEMS Porting Guide

L'installation d'un environnement de développement adapté a RTEMS sous Windows a été impossible. L'utilisation de minGW (minimalist GNU for Windows) et de MSYS (Minimal SYStem, un shell de commande de type unix) s'est révélée problématique du fait de l'interaction avec Cygwin qui est installée sur le poste de travail mis à disposition par l'école. Joel Sherill, l'une des fondateurs de RTEMS, a vivement conseillé dans un de ses mails à installer une version de Fedora13 (système d'exploitation basé sur UNIX) virtuelle à l'aide de VirtualBox. Cependant la mise en place de tous les outils nécessaire sur un OS vierge est longue et compliquée. Finalement une version existante de RTEMS (le portage pour la Nintendo DS utilisant un cœur ARM9TDMI) a réussi a finalement été compilé depuis le code source sans erreurs dans fedora13.

La compilation de RTEMS se fait par des logiciels automatiques, automake et autoconf qui génèrent tous les fichiers makefile nécessaire à la compilation de RTEMS à partir de fichiers partiels répartis dans tous les répertoires de RTEMS en fonction des paramètres passés sur la ligne de commande install. La documentation et l'utilisation de ces programmes est restée totalement opaque.

La compilation de la version pour la Nintendo DS a pris 45 minutes sur le système virtuel, n'incitant pas à l'exploration et aux essais de modification de code…

Tous ces problèmes mis à part, un début de portage a été réalisé. Comme conseillé par la documentation, la base de départ pour un portage est le répertoire rtems-4.9.4\cpukit\score\cpu\no_cpu contenant les fichiers de base qui doivent être implémentés. Ces fichiers sont les bases du SuperCore.

4.7.1 Portage blocs SuperCore et SuperCore CPU

Le portage de tout OS commence toujours par le codage des outils de bases que sont la gestion des interruptions, les switching contexts et la manipulation des registres dédiés à une application.

Le fichier cpu.h du répertoire rtems-4.9.4\cpukit\score\cpu\no_cpu\rtems\score définit toute une série de constantes qu'il faut configurer en fonction de l'architecture du processeur. Toutes les définitions ont été expliquées par des commentaires justifiant le choix. Ce fichier est disponible en annexe XXXI. Il y est décidé par exemple si le processeur gère plusieurs stacks en hardware, ce qui est le cas du Cortex-M3. Le stack par default est appelé main_stack, l'autre stack optionnel et utilisable uniquement en mode Thread est le process_stack. Le Cortex-M3 travaille dans 2 modes différents :

- Handler mode : lors du traitement d'un évènement ou d'une interruption
- Thread mode : application standard

La croissance du stack ainsi que les alignements mémoire que le code doit respecter sont aussi définis ici.

Concernant les switching contexts, une structure doit être déclarée qui contient les registres que le processeur ne sauve pas automatiquement lors d'une interruption. Le Cortex-M3 sauve les registres r0-r3, r12, sp_process lr et pc sur le sp_process. Il faut donc créer une structure comprenant les registres r4-r11. Une structure identique est aussi définie pour les entrées en interruption.

Vient ensuite la définition d'une enum qui contient tous les Handler des différentes interruptions.

Des routines de bas niveau en assembler ont été écrites pour activer, désactiver et flasher (activer et désactiver rapidement) les interruptions du Cortex-M3.

Le fichier cpu.c (AnnexeXXXII) implémente une fonction qui permet d'attribuer un nouvel Handler à une interruption et une autre qui crée la table de vecteur.

Le fichier asm.h (Annexe XXXII) a été modifié pour correspondre aux registres présents dans le Cortex-M3.

Le manque de temps n'as pas permis d'aller plus loin dans le portage de l'OS. Il reste cependant une bonne partie du travail à réaliser. Les drivers pour tous les périphériques internes du processeur STM32F103ZET6 ainsi que ceux gérant le hardware présents sur la carte doivent être écrits. Principalement, le driver de clock qui génèrera les ticks (intervalles de temps utilisés par la gestion du kernel) et un driver de timer hardware permettant à l'OS de gérer des compteurs software.

Cet OS gère nativement les systèmes multi processeurs et les mémoires dual-port. Le portage de cet OS sur la carte devrait permettre d'obtenir un système très performant…

5 Système

5.1 Consommation électrique

La consommation électrique minimale constatée pour la carte de démonstration est de 45mA toutes leds éteintes sauf la led témoin d'alimentation, les deux processeurs étant dans une boucle infinie vide. Cela correspond à la valeur donnée dans le tableau 18 du datasheet STM32F103ZET6.pdf dans la case 72MHz/All peripherals disabled de 30,5 mA par processeur. Le total serait donc de 63mA en comptant les 2mA de la led témoin. La différence avec la mesure vient probablement du fait que l'exécution d'une boucle vide infinie consomme moins d'énergie que du code normal.

La mesure la plus haute peut être mesurée lors de l'exécution de l'algorithme calculant les décimales de π avec les deux processeurs. Un courant de 150mA a été constaté.

Allumer toutes les leds et exécuter une boucle vide infinie sur les deux processeurs consomme aussi 150mA. Cela montre que la consommation du système est surtout due au leds, qui devraient demander 20 mA mais qui semblent consommer moins de 15mA chacune.

5.2 Performances applications multiprocesseur

Comme expliqué dans l'introduction, l'efficacité d'utilisation des ressources offertes par le système dépends surtout du logiciel et de son implémentation. La Figure 34 tirée d'un cite reconnu de matériel informatique montre bien cette dépendance. Une application est testée sur un processeur à 6 cœurs en activant les cœurs l'un après l'autre et en effectuant un benchmark. La même application est exécutée mais à gauche elle n'est pas du tout optimisée pour utiliser plusieurs cœurs alors qu'à droite elle est au contraire parfaitement optimisée pour utiliser au maximum les ressources du système. A gauche les performances stagnent tandis qu'à droite chaque cœur ajouté fournit le même gain absolu.

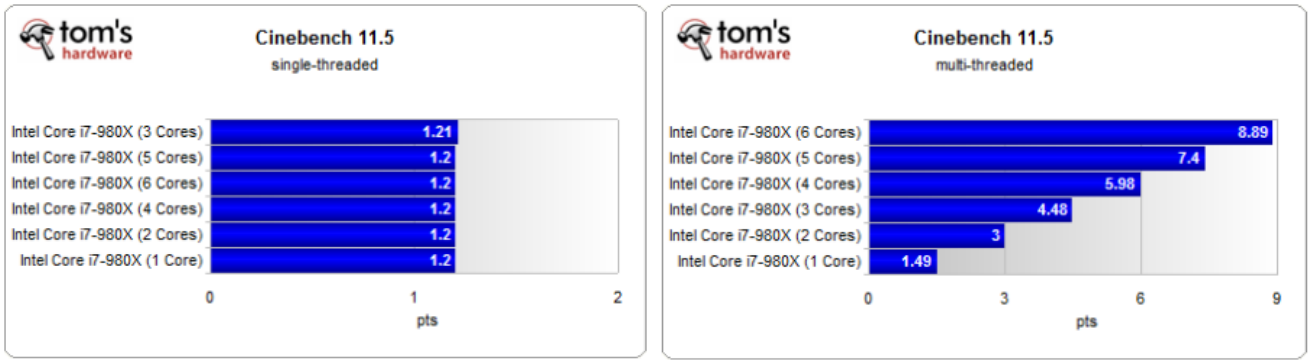


Figure 34 : Optimisation du software

Voici un bref aperçu des gains constatés avec l'application de calcul des décimales de π.

En mesurant le temps nécessaire à obtenir la table des 508 premières décimales avec un puis deux processeurs utilisés, une idée du gain obtenu et de l'overhead de parallélisation ajouté peut être extrapolée. Si le temps l'avait permis, l'horloge temps réel interne du processeur 1 aurait pu fournir des valeurs très exactes de la durée de chacune des tâches. Un simple chronomètre suffit toutefois à constater le gain obtenu.

Pour compléter la tâche avec un seul processeur actif, une moyenne de 11,5 secondes a été mesurée. La même tâche avec les deux processeurs actifs requiert une moyenne de 6 secondes. En réalisant la mesure sur une valeur plus grande de décimales de π, il est possible de diminuer l'erreur de mesure. En modifiant la méthode de passage des ordres entre le processeur 1 et le processeur 2 il serait possible de faire calculer jusqu'à 32768 par processeur. La mesure serait alors très précise.

Le gain constaté est donc bien une multiplication par 2 des performances. L'overhead peut être quantifié à à peut près 0.25 s (6 – (11.5/2)), ce qui est minime en comparaison avec le temps de traitement effectif. Certaines fois, rendre une application parallélisable demande tellement d'overhead que tout le gain que pourrait apporter la parallélisation disparaît à cause de l'ajout de code. Un exemple très parlant de ce cas de figure est visible en Figure **35**. Le passage de 1 à 2 cœurs apporte un léger gain mais tout cœur supplémentaire entraine une baisse des performances dues à soit à l'overhead soit aux switching contexts.

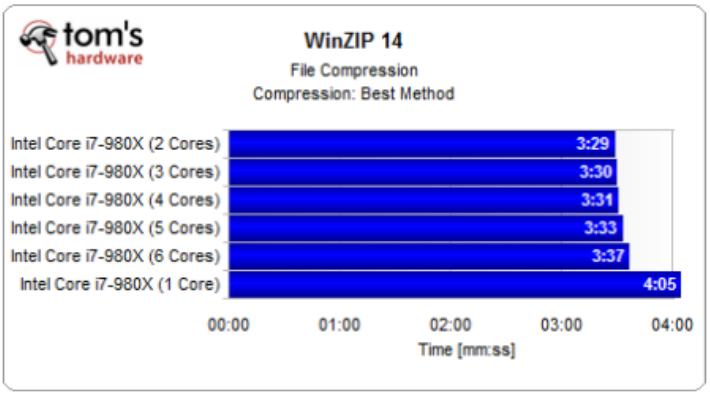


Figure 35 : Overhead trop important

Ce gain quasi idéal de 2 était prévisible dès le départ vu la très forte indépendance des tâches des deux processeurs. Dans la plus part des applications, les échanges et la synchronisation entre les processeurs sont beaucoup plus forts. Les gains en général constatés sont alors plutôt de 1,5 à 1,6 pour l'ajout d'un second cœur, avec un gain décroissant pour chaque cœur supplémentaire. Voici un exemple qui illustre le cas le plus courrant (Figure 36) :

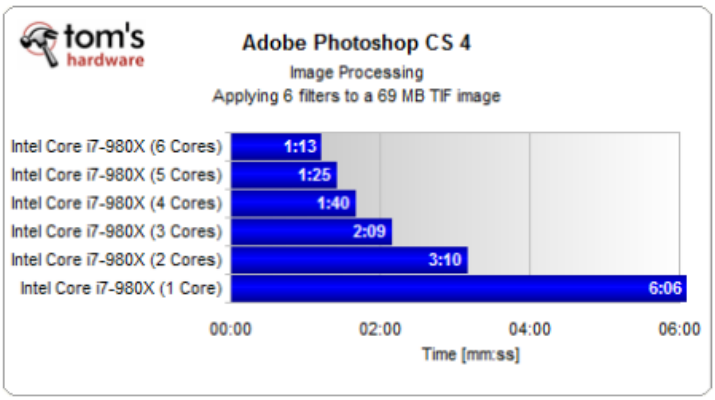


Figure 36 : Cas de gain le plus courant

6 Conclusions et perspectives

Le développement matériel de la carte n'a pas posé de difficultés majeures. Un excellent choix pour l'architecture mémoire du système ainsi qu'un modèle de processeur utilisé offrant des périphériques et des contrôleurs efficaces et entièrement configurables ont largement contribué à la facilité de conception du système. Il est intéressant de noter que le gestionnaire d'alimentation a été largement surdimensionné (800mA) au vu des 150mA de consommation maximale observée. Les périphériques USB ainsi que l'utilisation de l'horloge temps réel n'ont pas été testés et pourraient faire l'objet de développements futurs.

La compréhension des mécanismes complexes cachés par un IDE configuré et fonctionnel a permis d'acquérir une bonne expérience du cross-developpement et du remote debugging. L'aide initiale apportée par le projet NTRT a permis d'appréhender les bases de ces techniques. La configuration d'OpenOCD et de GDB est l'étape fondamentale pour constituer une toolchain fonctionnelle.

La présence de 2 processeurs sur le système force l'utilisation de deux instances d'Eclipse. OpenOCD est capable de gérér une chaine JTAG sérialisée mais GDB et Eclipse ne sont pas en mesure de debugger deux application simultanément, c'est pourquoi la chaine JTAG est restée en parallèle. La question de la simplification de debuggage d'un système multi processeur est donc restée sans réponse. Mais la méthode utilisée dans le cadre de ce travail de diplôme a tout de même permis de développer quelques applications basiques.

Le développement de l'application de calcul des décimales de π a dirigé la réflexion sur l'aspect logiciel de la programmation parallèle et de ses contraintes et limites. Développer cet algorithme a montré combien il est fondamental de passer par une étape d'étude, de schématisation et de planification des interactions entre les deux processeurs pour arriver à obtenir un gain de performances appréciable. Bien que la programmation séquentielle exige aussi ces études préalables elles sont d'autant plus importantes dans la conception d'une application parallèle. Le type d'application à développer conditionne complètement les parties qui sont parralélisables.

L'objectif le plus intéressant de ce travail de diplôme, le portage de l'OS Rtems n'a malheuresement pas pu être mené à terme. La difficulté d'installer un environnement de développement stable et fonctionnel, la complexité de ses outils automatiques de compilation et sa structure, qui en schématique parait claire mais qui dans la réalité est quasiment différente pour chaque processeur, ont empêché d'avancer assez vite pour terminer le portage. Le SuperCore est en grande partie porté bien qu'il n'ait pas pu être testé. Avec du temps supplémentaire et une meilleure connaissance des systèmes de type UNIX la réalisation de cet objectif pourrait être atteinte. Le portage de microOS laissé en option n'a pas été abordé car des portages fonctionnels sont déjà utilisables et il semblait peu utile de refaire ce travail.

Ce travail de diplôme très enthousiasmant et vraiment en phase avec les nouvelles technologies aura sans aucun doute permis de d'acquérir de l'expérience dans un domaine en pleine émergence.

Sion, le 12 Juillet 2010 Christian Castellaro

Références

STM :

[RM0008] Reference Manual : STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM-based 32-bit MCUs.

STM32F103xC STM32F103xD STM32F103xE High-density performance line ARM-based 32-bit MCU with 256 to512KB Flash, USB, CAN, 11 timers, 3 ADCs, 13 communication interfaces

ARM

Cortex™-M3 Technical Reference Manual Revision r2p0 Copyright © 2005-2008, 2010 ARM Limited. All rights reserved.

Wikipedia

http://en.wikipedia.org/wiki/Bailey-Borwein-Plouffe_formula

http://en.wikipedia.org/wiki/Spigot_algorithm

http://en.wikipedia.org/wiki/Modular_exponentiation

Joseph Yiu, The Definitive Guide to the ARM Cortex-M3 (Embedded Technology Series)

Annexes

Annexe I	:	Liste complète des composants
Annexe II	:	Estimation mémoire vive pour RTEMS
Annexe III	:	Pining processeur détaillé
Annexe IV	:	Pining mémoire
Annexe V	:	Machine d'état TAP
Annexe VI	:	Estimation de la consommation maximale de la carte
Annexe VII	:	Schématique découplage processeur 1 + oscill. 8Mhz
Annexe VIII	:	Schématique périphériques + quartz 32.768 KHz processeur1
Annexe IX	:	Schématique découplage processeur 2
Annexe X	:	Schématique périphériques + quartz 32.768 KHz processeur2
Annexe XI	:	Schématique mémoire
Annexe XII	:	Schématique JTAG
Annexe XIII	:	Schématique reset
Annexe XIV	:	Schématique I/O
Annexe XV	:	Schématique power
Annexe XVI	:	Schématique USB
Annexe XVII	:	Routage top
Annexe XVIII	:	Routage bottom
Annexe XIX	:	Routage Vcc
Annexe XX	:	Routage Gnd
Annexe XXI	:	Commande farnell + références au datasheets
Annexe XXII	:	clock tree
Annexe XXIII	:	RTC_RCC_config.c et .h
Annexe XXIV	:	GPIO_config.c et .h
Annexe XXV	:	FSMC_config.c et .h
Annexe XXVI	:	EXTI_config.c et .h
Annexe XXVII	:	NVIC_config.c et .h
Annexe XXVIII	:	test_app.c, app_int.h et .c
Annexe XXIX	:	app1.c
Annexe XXX	:	app2.c
Annexe XXXI	:	cpu.h
Annexe XXXII	:	cpu.c
Annexe XXXIII	:	asm.h

Annexe II : estimation mémoire pour utiliser Rtems

RTEMS RAM Workspace Worksheet			
Description	**Quantity**	**Need**	**Bytes Required**
maximum_tasks	3	400	1200
maximum_timers	2	68	136
maximum_semaphores	2	124	248
maximum_message_queues	2	148	296
maximum_regions		144	0
maximum_partitions		56	0
maximum_ports		36	0
maximum_periods		36	0
maximum_extensions		64	0
Floating Point Tasks		332	0
Task Stacks	3	200	600
Total Single Processor Requirements			**2480**
Description		**Equation**	**Bytes Required**
maximum_nodes	2	48	96
maximum_global_objects	2	20	40
maximum_proxies		124	
Total Multiprocessing Requirements			**136**
Fixed System Requirements			**8872**
Total Single Processor Requirements			2480
Total Multiprocessing Requirements			136
Minimum Bytes for RTEMS Workspace			**11488**

All units are in bits

Annexe III : Pining détaillé du processeur STM32F103ZET6

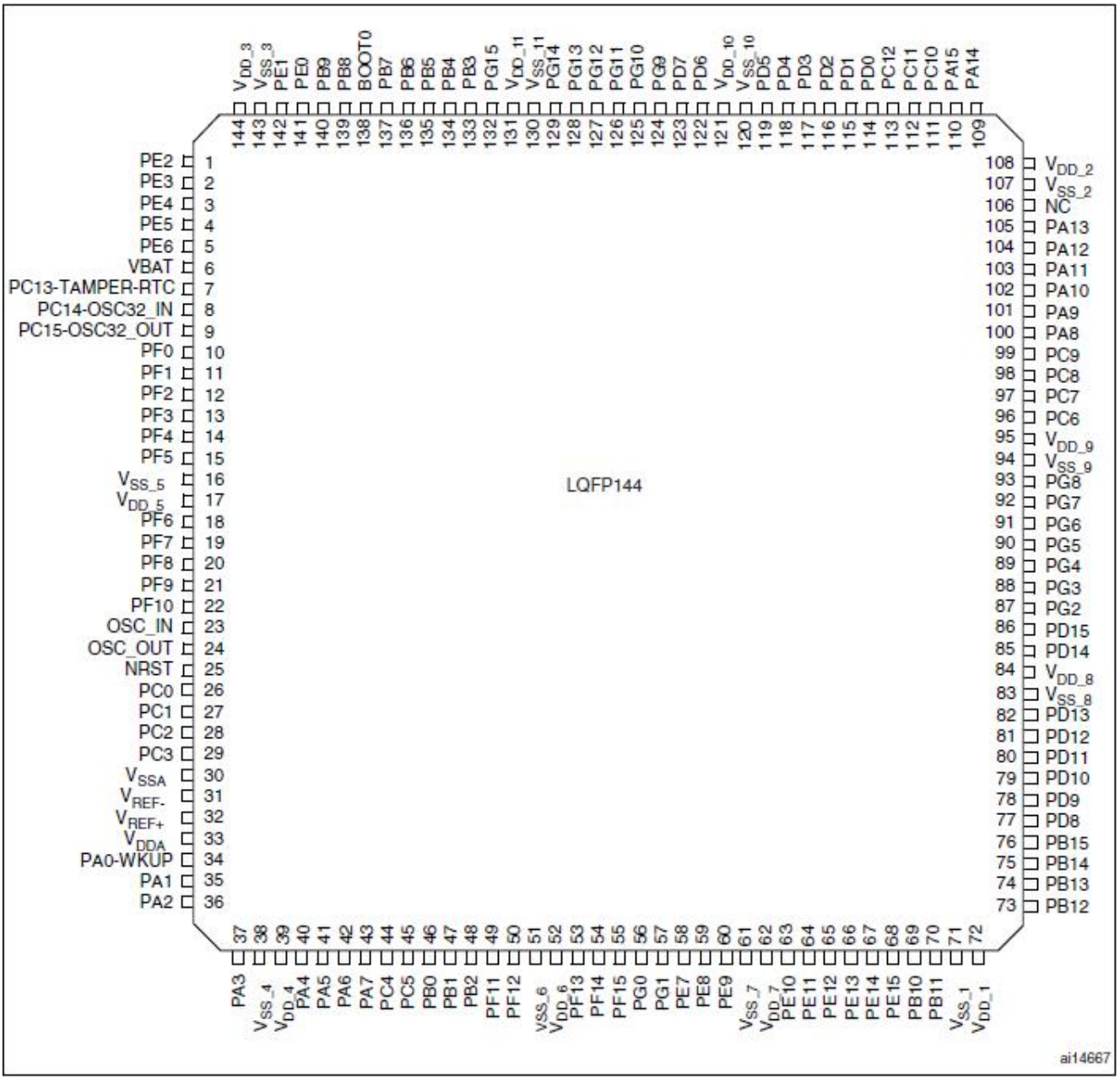


Table 5. High-density STM32F103xx pin definitions

Pins						Pin name	Type(1)	I / O Level(2)	Main function(3) (after reset)	Alternate functions(4)	
LFBGA144	LFBGA100	WLCSP64	LQFP64	LQFP100	LQFP144					Default	Remap
A3	A3	-	-	1	1	PE2	I/O	FT	PE2	TRACECK/ FSMC_A23	
A2	B3	-	-	2	2	PE3	I/O	FT	PE3	TRACED0/FSMC_A19	
B2	C3	-	-	3	3	PE4	I/O	FT	PE4	TRACED1/FSMC_A20	
B3	D3	-	-	4	4	PE5	I/O	FT	PE5	TRACED2/FSMC_A21	
B4	E3	-	-	5	5	PE6	I/O	FT	PE6	TRACED3/FSMC_A22	
C2	B2	C6	1	6	6	V_{BAT}	S		V_{BAT}		
A1	A2	C8	2	7	7	PC13-TAMPER-RTC(5)	I/O		PC13(6)	TAMPER-RTC	
B1	A1	B8	3	8	8	PC14-OSC32_IN(5)	I/O		PC14(6)	OSC32_IN	
C1	B1	B7	4	9	9	PC15-OSC32_OUT(5)	I/O		PC15(6)	OSC32_OUT	
C3	-	-	-	-	10	PF0	I/O	FT	PF0	FSMC_A0	
C4	-	-	-	-	11	PF1	I/O	FT	PF1	FSMC_A1	
D4	-	-	-	-	12	PF2	I/O	FT	PF2	FSMC_A2	
E2	-	-	-	-	13	PF3	I/O	FT	PF3	FSMC_A3	
E3	-	-	-	-	14	PF4	I/O	FT	PF4	FSMC_A4	
E4	-	-	-	-	15	PF5	I/O	FT	PF5	FSMC_A5	
D2	C2	-	-	10	16	V_{SS_5}	S		V_{SS_5}		
D3	D2	-	-	11	17	V_{DD_5}	S		V_{DD_5}		
F3	-	-	-	-	18	PF6	I/O		PF6	ADC3_IN4/FSMC_NIORD	
F2	-	-	-	-	19	PF7	I/O		PF7	ADC3_IN5/FSMC_NREG	
G3	-	-	-	-	20	PF8	I/O		PF8	ADC3_IN6/FSMC_NIOWR	
G2	-	-	-	-	21	PF9	I/O		PF9	ADC3_IN7/FSMC_CD	
G1	-	-	-	-	22	PF10	I/O		PF10	ADC3_IN8/FSMC_INTR	
D1	C1	D8	5	12	23	OSC_IN	I		OSC_IN		
E1	D1	D7	6	13	24	OSC_OUT	O		OSC_OUT		
F1	E1	C7	7	14	25	NRST	I/O		NRST		
H1	F1	E8	8	15	26	PC0	I/O		PC0	ADC123_IN10	
H2	F2	F8	9	16	27	PC1	I/O		PC1	ADC123_IN11	
H3	E2	D6	10	17	28	PC2	I/O		PC2	ADC123_IN12	
H4	F3	-	11	18	29	PC3	I/O		PC3	ADC123_IN13	
J1	G1	E7	12	19	30	V_{SSA}	S		V_{SSA}		

Table 5. High-density STM32F103xx pin definitions (continued)

Pins						Pin name	Type(1)	I / O Level(2)	Main function(3) (after reset)	Alternate functions(4)	
LFBGA144	LFBGA100	WLCSP64	LQFP64	LQFP100	LQFP144					Default	Remap
K1	H1	-	-	20	31	V_{REF-}	S		V_{REF-}		
L1	J1	F7 (7)	-	21	32	V_{REF+}	S		V_{REF+}		
M1	K1	G8	13	22	33	V_{DDA}	S		V_{DDA}		
J2	G2	F6	14	23	34	PA0-WKUP	I/O		PA0	WKUP/USART2_CTS(8) ADC123_IN0 TIM2_CH1_ETR TIM5_CH1/TIM8_ETR	
K2	H2	E6	15	24	35	PA1	I/O		PA1	USART2_RTS(8) ADC123_IN1/ TIM5_CH2/TIM2_CH2(8)	
L2	J2	H8	16	25	36	PA2	I/O		PA2	USART2_TX(8)/TIM5_CH3 ADC123_IN2/ TIM2_CH3 (8)	
M2	K2	G7	17	26	37	PA3	I/O		PA3	USART2_RX(8)/TIM5_CH4 ADC123_IN3/TIM2_CH4(8)	
G4	E4	F5	18	27	38	V_{SS_4}	S		V_{SS_4}		
F4	F4	G6	19	28	39	V_{DD_4}	S		V_{DD_4}		
J3	G3	H7	20	29	40	PA4	I/O		PA4	SPI1_NSS(8)/ USART2_CK(8) DAC_OUT1/ADC12_IN4	
K3	H3	E5	21	30	41	PA5	I/O		PA5	SPI1_SCK(8) DAC_OUT2 ADC12_IN5	
L3	J3	G5	22	31	42	PA6	I/O		PA6	SPI1_MISO(8) TIM8_BKIN/ADC12_IN6 TIM3_CH1(8)	TIM1_BKIN
M3	K3	G4	23	32	43	PA7	I/O		PA7	SPI1_MOSI(8)/ TIM8_CH1N/ADC12_IN7 TIM3_CH2(8)	TIM1_CH1N
J4	G4	H6	24	33	44	PC4	I/O		PC4	ADC12_IN14	
K4	H4	H5	25	34	45	PC5	I/O		PC5	ADC12_IN15	
L4	J4	H4	26	35	46	PB0	I/O		PB0	ADC12_IN8/TIM3_CH3 TIM8_CH2N	TIM1_CH2N
M4	K4	F4	27	36	47	PB1	I/O		PB1	ADC12_IN9/TIM3_CH4(8) TIM8_CH3N	TIM1_CH3N
J5	G5	H3	28	37	48	PB2	I/O	FT	PB2/BOOT1		
M5	-	-	-	-	49	PF11	I/O	FT	PF11	FSMC_NIOS16	
L5	-	-	-	-	50	PF12	I/O	FT	PF12	FSMC_A6	

Table 5. High-density STM32F103xx pin definitions (continued)

Pins						Pin name	Type(1)	I / O Level(2)	Main function(3) (after reset)	Alternate functions(4)	
LFBGA144	LFBGA100	WLCSP64	LQFP64	LQFP100	LQFP144					Default	Remap
H5	-	-	-	-	51	V_{SS_6}	S		V_{SS_6}		
G5	-	-	-	-	52	V_{DD_6}	S		V_{DD_6}		
K5	-	-	-	-	53	PF13	I/O	FT	PF13	FSMC_A7	
M6	-	-	-	-	54	PF14	I/O	FT	PF14	FSMC_A8	
L6	-	-	-	-	55	PF15	I/O	FT	PF15	FSMC_A9	
K6	-	-	-	-	56	PG0	I/O	FT	PG0	FSMC_A10	
J6	-	-	-	-	57	PG1	I/O	FT	PG1	FSMC_A11	
M7	H5	-	-	38	58	PE7	I/O	FT	PE7	FSMC_D4	TIM1_ETR
L7	J5	-	-	39	59	PE8	I/O	FT	PE8	FSMC_D5	TIM1_CH1N
K7	K5	-	-	40	60	PE9	I/O	FT	PE9	FSMC_D6	TIM1_CH1
H6	-	-	-	-	61	V_{SS_7}	S		V_{SS_7}		
G6	-	-	-	-	62	V_{DD_7}	S		V_{DD_7}		
J7	G6	-	-	41	63	PE10	I/O	FT	PE10	FSMC_D7	TIM1_CH2N
H8	H6	-	-	42	64	PE11	I/O	FT	PE11	FSMC_D8	TIM1_CH2
J8	J6	-	-	43	65	PE12	I/O	FT	PE12	FSMC_D9	TIM1_CH3N
K8	K6	-	-	44	66	PE13	I/O	FT	PE13	FSMC_D10	TIM1_CH3
L8	G7	-	-	45	67	PE14	I/O	FT	PE14	FSMC_D11	TIM1_CH4
M8	H7	-	-	46	68	PE15	I/O	FT	PE15	FSMC_D12	TIM1_BKIN
M9	J7	G3	29	47	69	PB10	I/O	FT	PB10	I2C2_SCL/USART3_TX(8)	TIM2_CH3
M10	K7	F3	30	48	70	PB11	I/O	FT	PB11	I2C2_SDA/USART3_RX(8)	TIM2_CH4
H7	E7	H2	31	49	71	V_{SS_1}	S		V_{SS_1}		
G7	F7	H1	32	50	72	V_{DD_1}	S		V_{DD_1}		
M11	K8	G2	33	51	73	PB12	I/O	FT	PB12	SPI2_NSS/I2S2_WS/ I2C2_SMBA/ USART3_CK(8)/ TIM1_BKIN(8)	
M12	J8	G1	34	52	74	PB13	I/O	FT	PB13	SPI2_SCK/I2S2_CK USART3_CTS(8)/ TIM1_CH1N	
L11	H8	F2	35	53	75	PB14	I/O	FT	PB14	SPI2_MISO/TIM1_CH2N USART3_RTS(8)/	
L12	G8	F1	36	54	76	PB15	I/O	FT	PB15	SPI2_MOSI/I2S2_SD TIM1_CH3N(8)/	
L9	K9	-	-	55	77	PD8	I/O	FT	PD8	FSMC_D13	USART3_TX
K9	J9	-	-	56	78	PD9	I/O	FT	PD9	FSMC_D14	USART3_RX

Table 5. High-density STM32F103xx pin definitions (continued)

Pins						Pin name	Type(1)	I / O Level(2)	Main function(3) (after reset)	Alternate functions(4)	
LFBGA144	LFBGA100	WLCSP64	LQFP64	LQFP100	LQFP144					Default	Remap
J9	H9	-	-	57	79	PD10	I/O	FT	PD10	FSMC_D15	USART3_CK
H9	G9	-	-	58	80	PD11	I/O	FT	PD11	FSMC_A16	USART3_CTS
L10	K10	-	-	59	81	PD12	I/O	FT	PD12	FSMC_A17	TIM4_CH1 / USART3_RTS
K10	J10	-	-	60	82	PD13	I/O	FT	PD13	FSMC_A18	TIM4_CH2
G8	-	-	-	-	83	V_{SS_8}	S		V_{SS_8}		
F8	-	-	-	-	84	V_{DD_8}	S		V_{DD_8}		
K11	H10	-	-	61	85	PD14	I/O	FT	PD14	FSMC_D0	TIM4_CH3
K12	G10	-	-	62	86	PD15	I/O	FT	PD15	FSMC_D1	TIM4_CH4
J12	-	-	-	-	87	PG2	I/O	FT	PG2	FSMC_A12	
J11	-	-	-	-	88	PG3	I/O	FT	PG3	FSMC_A13	
J10	-	-	-	-	89	PG4	I/O	FT	PG4	FSMC_A14	
H12	-	-	-	-	90	PG5	I/O	FT	PG5	FSMC_A15	
H11	-	-	-	-	91	PG6	I/O	FT	PG6	FSMC_INT2	
H10	-	-	-	-	92	PG7	I/O	FT	PG7	FSMC_INT3	
G11	-	-	-	-	93	PG8	I/O	FT	PG8		
G10	-	-	-	-	94	V_{SS_9}	S		V_{SS_9}		
F10	-	-	-	-	95	V_{DD_9}	S		V_{DD_9}		
G12	F10	E1	37	63	96	PC6	I/O	FT	PC6	I2S2_MCK/ TIM8_CH1/SDIO_D6	TIM3_CH1
F12	E10	E2	38	64	97	PC7	I/O	FT	PC7	I2S3_MCK/ TIM8_CH2/SDIO_D7	TIM3_CH2
F11	F9	E3	39	65	98	PC8	I/O	FT	PC8	TIM8_CH3/SDIO_D0	TIM3_CH3
E11	E9	D1	40	66	99	PC9	I/O	FT	PC9	TIM8_CH4/SDIO_D1	TIM3_CH4
E12	D9	E4	41	67	100	PA8	I/O	FT	PA8	USART1_CK/ TIM1_CH1(8)/MCO	
D12	C9	D2	42	68	101	PA9	I/O	FT	PA9	USART1_TX(8)/ TIM1_CH2(8)	
D11	D10	D3	43	69	102	PA10	I/O	FT	PA10	USART1_RX(8)/ TIM1_CH3(8)	
C12	C10	C1	44	70	103	PA11	I/O	FT	PA11	USART1_CTS/USBDM CAN_RX(8)/TIM1_CH4(8)	
B12	B10	C2	45	71	104	PA12	I/O	FT	PA12	USART1_RTS/USBDP/ CAN_TX(8)/TIM1_ETR(8)	

Table 5. High-density STM32F103xx pin definitions (continued)

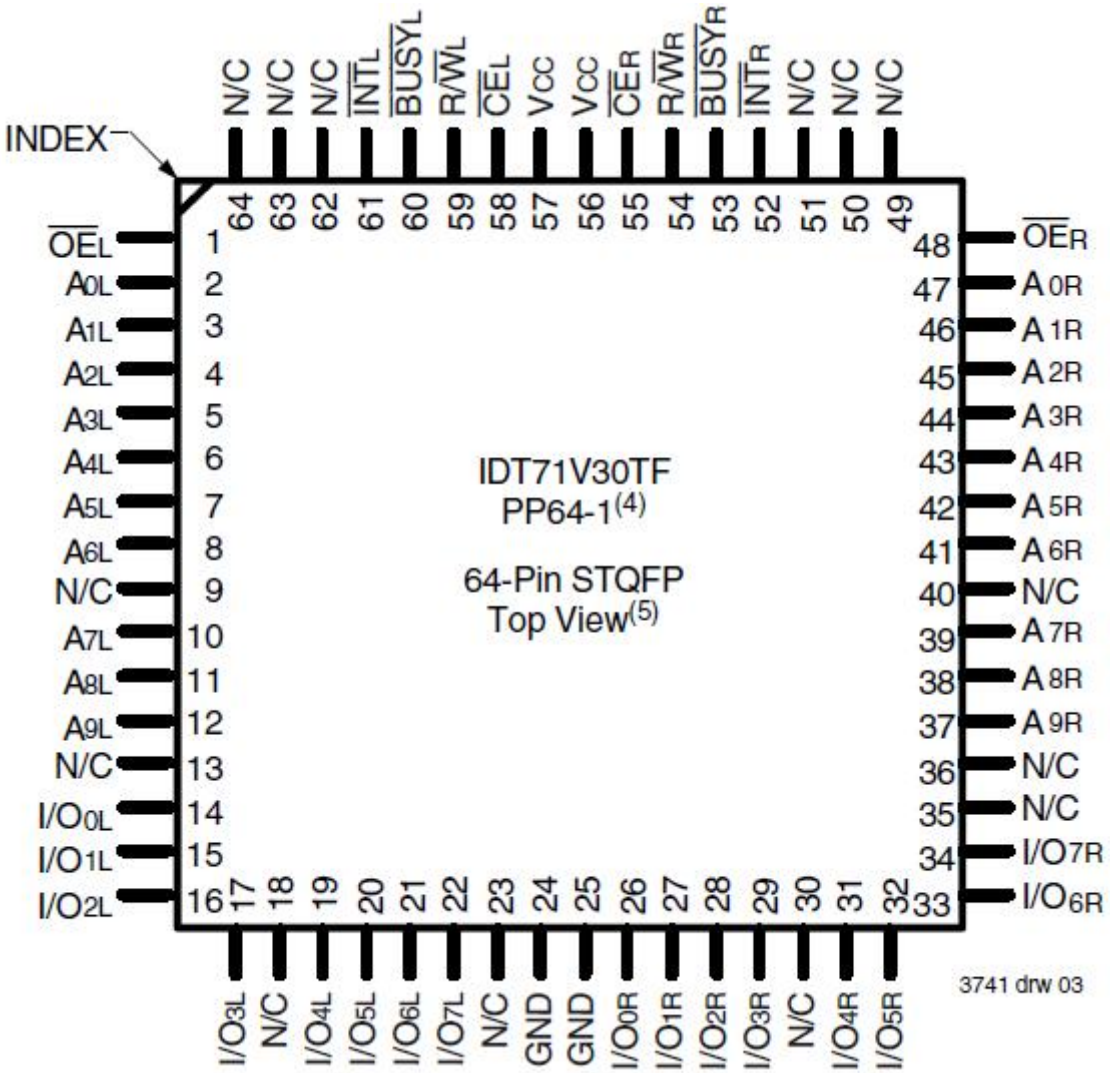
Pins						Pin name	Type(1)	I / O Level(2)	Main function(3) (after reset)	Alternate functions(4)	
LFBGA144	LFBGA100	WLCSP64	LQFP64	LQFP100	LQFP144					Default	Remap
A12	A10	D4	46	72	105	PA13	I/O	FT	JTMS-SWDIO		PA13
C11	F8	-	-	73	106	Not connected					
G9	E6	B1	47	74	107	V_{SS_2}	S		V_{SS_2}		
F9	F6	A1	48	75	108	V_{DD_2}	S		V_{DD_2}		
A11	A9	B2	49	76	109	PA14	I/O	FT	JTCK-SWCLK		PA14
A10	A8	C3	50	77	110	PA15	I/O	FT	JTDI	SPI3_NSS/ I2S3_WS	TIM2_CH1_ETR PA15 / SPI1_NSS
B11	B9	A2	51	78	111	PC10	I/O	FT	PC10	UART4_TX/SDIO_D2	USART3_TX
B10	B8	B3	52	79	112	PC11	I/O	FT	PC11	UART4_RX/SDIO_D3	USART3_RX
C10	C8	C4	53	80	113	PC12	I/O	FT	PC12	UART5_TX/SDIO_CK	USART3_CK
E10	D8	D8	5	81	114	PD0	I/O	FT	OSC_IN(9)	FSMC_D2(10)	CAN_RX
D10	E8	D7	6	82	115	PD1	I/O	FT	OSC_OUT(9)	FSMC_D3(10)	CAN_TX
E9	B7	A3	54	83	116	PD2	I/O	FT	PD2	TIM3_ETR/UART5_RX SDIO_CMD	
D9	C7	-	-	84	117	PD3	I/O	FT	PD3	FSMC_CLK	USART2_CTS
C9	D7	-	-	85	118	PD4	I/O	FT	PD4	FSMC_NOE	USART2_RTS
B9	B6	-	-	86	119	PD5	I/O	FT	PD5	FSMC_NWE	USART2_TX
E7	-	-	-	-	120	V_{SS_10}	S		V_{SS_10}		
F7	-	-	-	-	121	V_{DD_10}	S		V_{DD_10}		
A8	C6	-	-	87	122	PD6	I/O	FT	PD6	FSMC_NWAIT	USART2_RX
A9	D6	-	-	88	123	PD7	I/O	FT	PD7	FSMC_NE1/FSMC_NCE2	USART2_CK
E8	-	-	-	-	124	PG9	I/O	FT	PG9	FSMC_NE2/FSMC_NCE3	
D8	-	-	-	-	125	PG10	I/O	FT	PG10	FSMC_NCE4_1/ FSMC_NE3	
C8	-	-	-	-	126	PG11	I/O	FT	PG11	FSMC_NCE4_2	
B8	-	-	-	-	127	PG12	I/O	FT	PG12	FSMC_NE4	
D7	-	-	-	-	128	PG13	I/O	FT	PG13	FSMC_A24	
C7	-	-	-	-	129	PG14	I/O	FT	PG14	FSMC_A25	
E6	-	-	-	-	130	V_{SS_11}	S		V_{SS_11}		
F6	-	-	-	-	131	V_{DD_11}	S		V_{DD_11}		
B7	-	-	-	-	132	PG15	I/O	FT	PG15		

Table 5. High-density STM32F103xx pin definitions (continued)

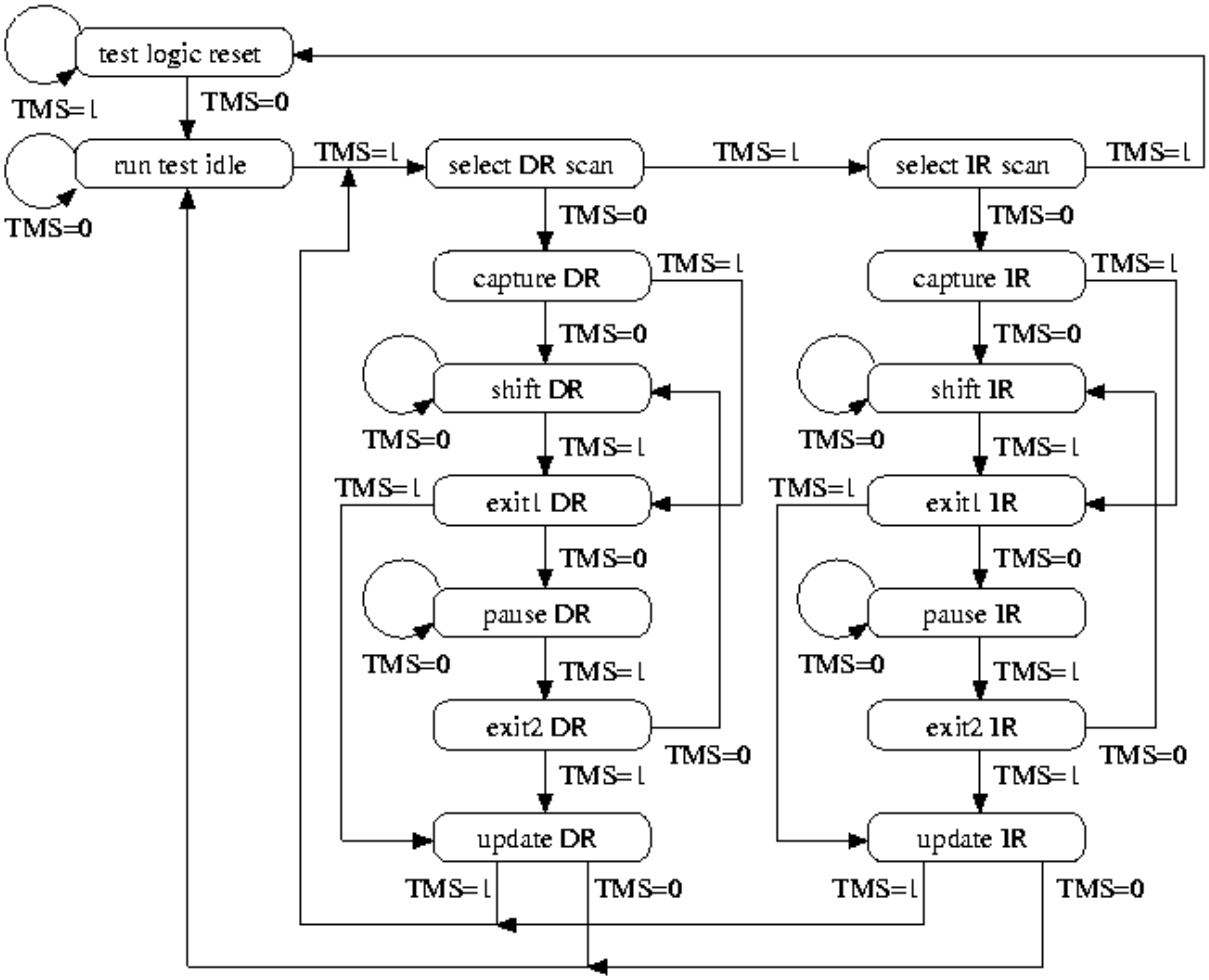
Pins						Pin name	Type(1)	I / O Level(2)	Main function(3) (after reset)	Alternate functions(4)	
LFBGA144	LFBGA100	WLCSP64	LQFP64	LQFP100	LQFP144					Default	Remap
A7	A7	A4	55	89	133	PB3/	I/O	FT	JTDO	SPI3_SCK / I2S3_CK/	PB3/TRACESWO TIM2_CH2 / SPI1_SCK
A6	A6	B4	56	90	134	PB4	I/O	FT	NJTRST	SPI3_MISO	PB4 / TIM3_CH1 SPI1_MISO
B6	C5	A5	57	91	135	PB5	I/O		PB5	I2C1_SMBA/ SPI3_MOSI I2S3_SD	TIM3_CH2 / SPI1_MOSI
C6	B5	B5	58	92	136	PB6	I/O	FT	PB6	I2C1_SCL(8)/ TIM4_CH1(8)	USART1_TX
D6	A5	C5	59	93	137	PB7	I/O	FT	PB7	I2C1_SDA(8) / FSMC_NADV / TIM4_CH2(8)	USART1_RX
D5	D5	A6	60	94	138	BOOT0	I		BOOT0		
C5	B4	D5	61	95	139	PB8	I/O	FT	PB8	TIM4_CH3(8)/SDIO_D4	I2C1_SCL/ CAN_RX
B5	A4	B6	62	96	140	PB9	I/O	FT	PB9	TIM4_CH4(8)/SDIO_D5	I2C1_SDA / CAN_TX
A5	D4	-	-	97	141	PE0	I/O	FT	PE0	TIM4_ETR / FSMC_NBL0	
A4	C4	-	-	98	142	PE1	I/O	FT	PE1	FSMC_NBL1	
E5	E5	A7	63	99	143	V_{SS_3}	S		V_{SS_3}		
F5	F5	A8	64	100	144	V_{DD_3}	S		V_{DD_3}		

1. I = input, O = output, S = supply.
2. FT = 5 V tolerant.
3. Function availability depends on the chosen device.
4. If several peripherals share the same I/O pin, to avoid conflict between these alternate functions only one peripheral should be enabled at a time through the peripheral clock enable bit (in the corresponding RCC peripheral clock enable register).
5. PC13, PC14 and PC15 are supplied through the power switch. Since the switch only sinks a limited amount of current (3 mA), the use of GPIOs PC13 to PC15 in output mode is limited: the speed should not exceed 2 MHz with a maximum load of 30 pF and these IOs must not be used as a current source (e.g. to drive an LED).
6. Main function after the first backup domain power-up. Later on, it depends on the contents of the Backup registers even after reset (because these registers are not reset by the main reset). For details on how to manage these IOs, refer to the Battery backup domain and BKP register description sections in the STM32F10xxx reference manual, available from the STMicroelectronics website: www.st.com.
7. Unlike in the LQFP64 package, there is no PC3 in the WLCSP package. The V_{REF+} functionality is provided instead.
8. This alternate function can be remapped by software to some other port pins (if available on the used package). For more details, refer to the Alternate function I/O and debug configuration section in the STM32F10xxx reference manual, available from the STMicroelectronics website: www.st.com.
9. For the LQFP64 package, the pins number 5 and 6 are configured as OSC_IN/OSC_OUT after reset, however the functionality of PD0 and PD1 can be remapped by software on these pins. For the LQFP100/BGA100 and LQFP144/BGA144 packages, PD0 and PD1 are available by default, so there is no need for remapping. For more details, refer to Alternate function I/O and debug configuration section in the STM32F10xxx reference manual.
10. For devices delivered in LQFP64 packages, the FSMC function is not available.

Annexe IV : pining de la mémoire



Annexe V : machine d'états du TAP

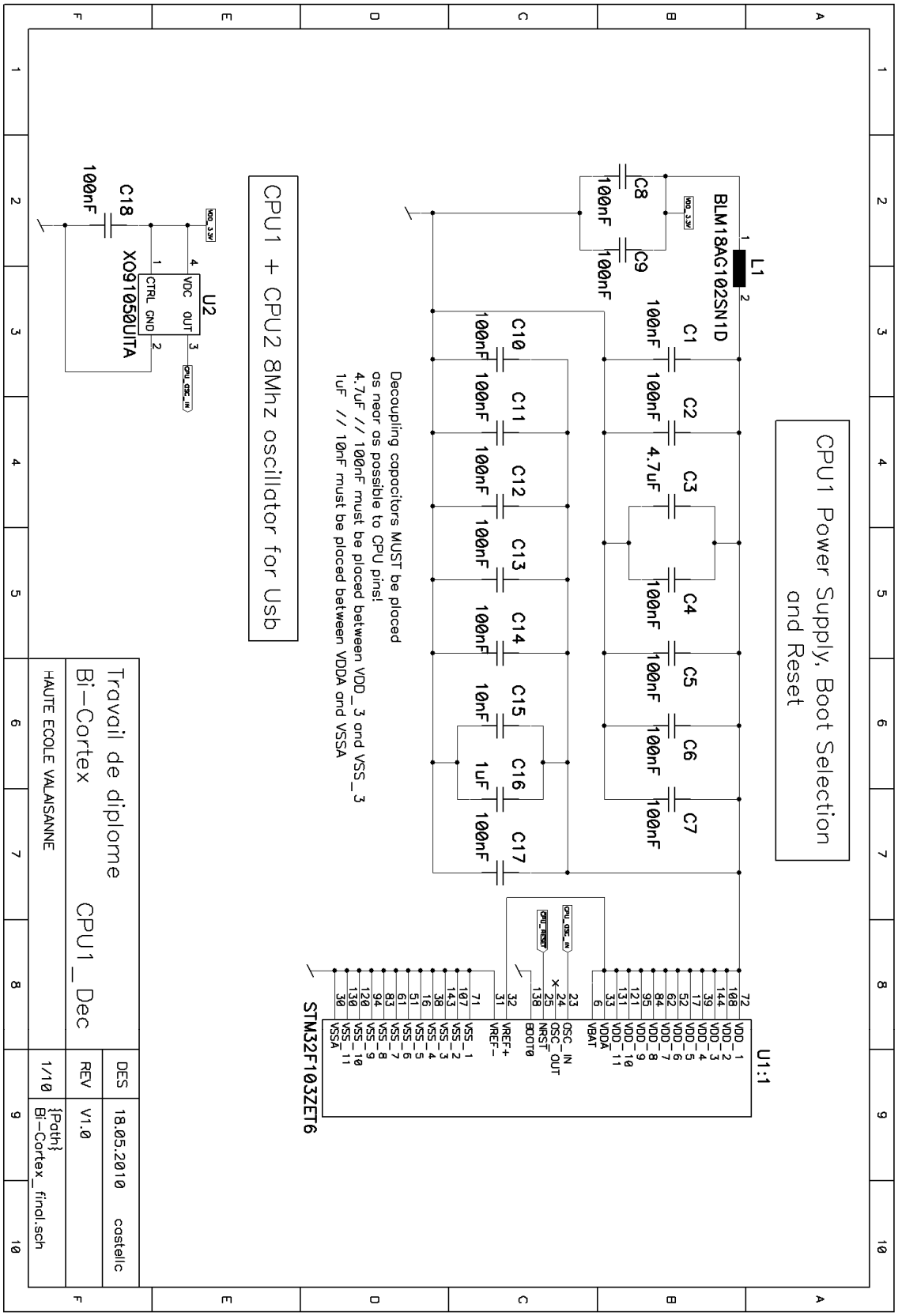


JTAG Test Access Port (TAP) controller state transition diagram

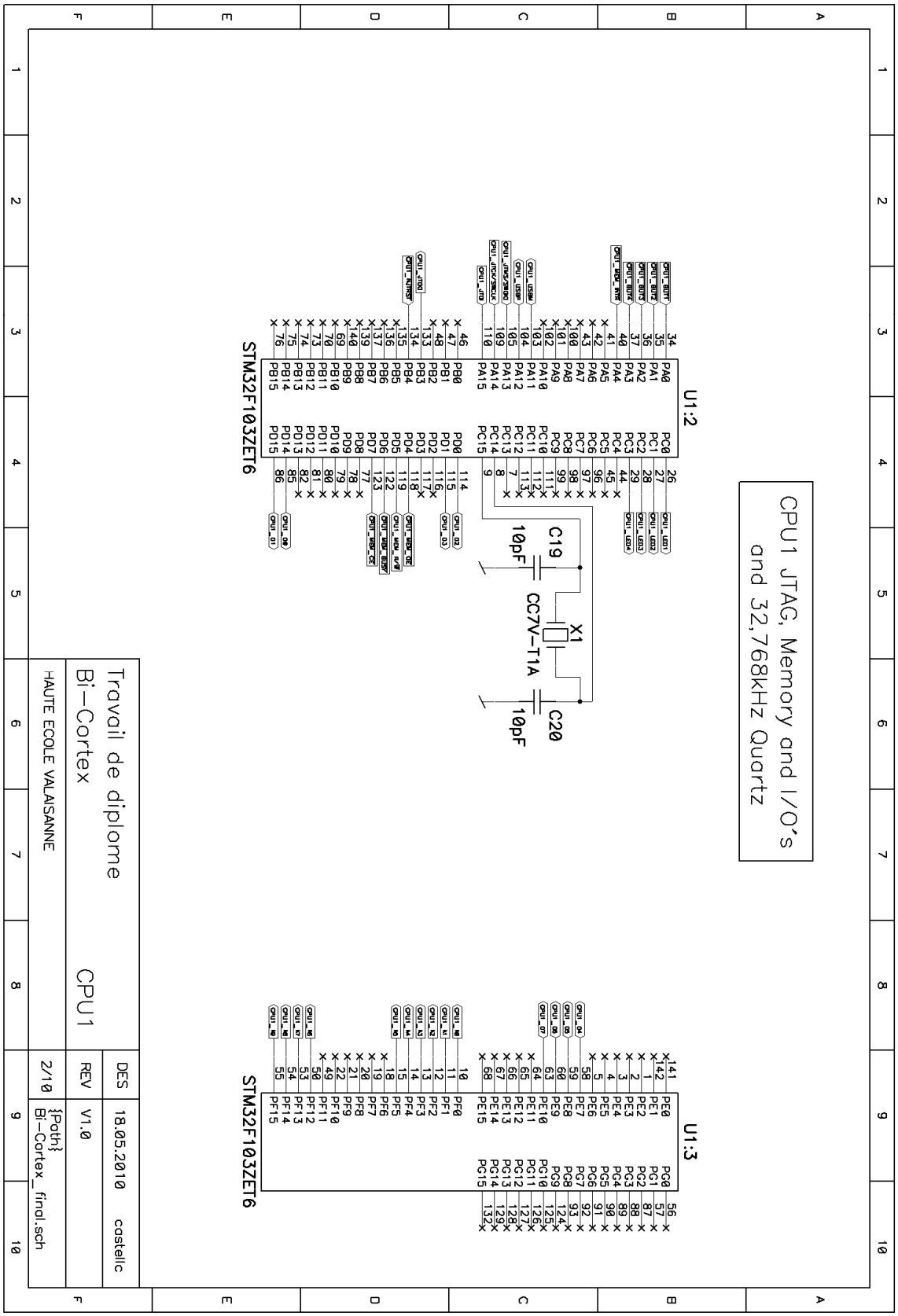
Annexe VI : estimation consommation max du système

Composant	Quantité	Consommation (mA)	Total part. (mA)
STM32F103ZET6 (processeur)	2	150	300
IDT71V30L25TFG (mémoire)	1	150	150
KP3216 SGD (leds feedback)	8	20	160
TLMO 1000 (led power)	1	2	2
NC7S08M5X (tiny gate AND)	4	12.5	50
marge	1	50	50
		TOTAL	**712**

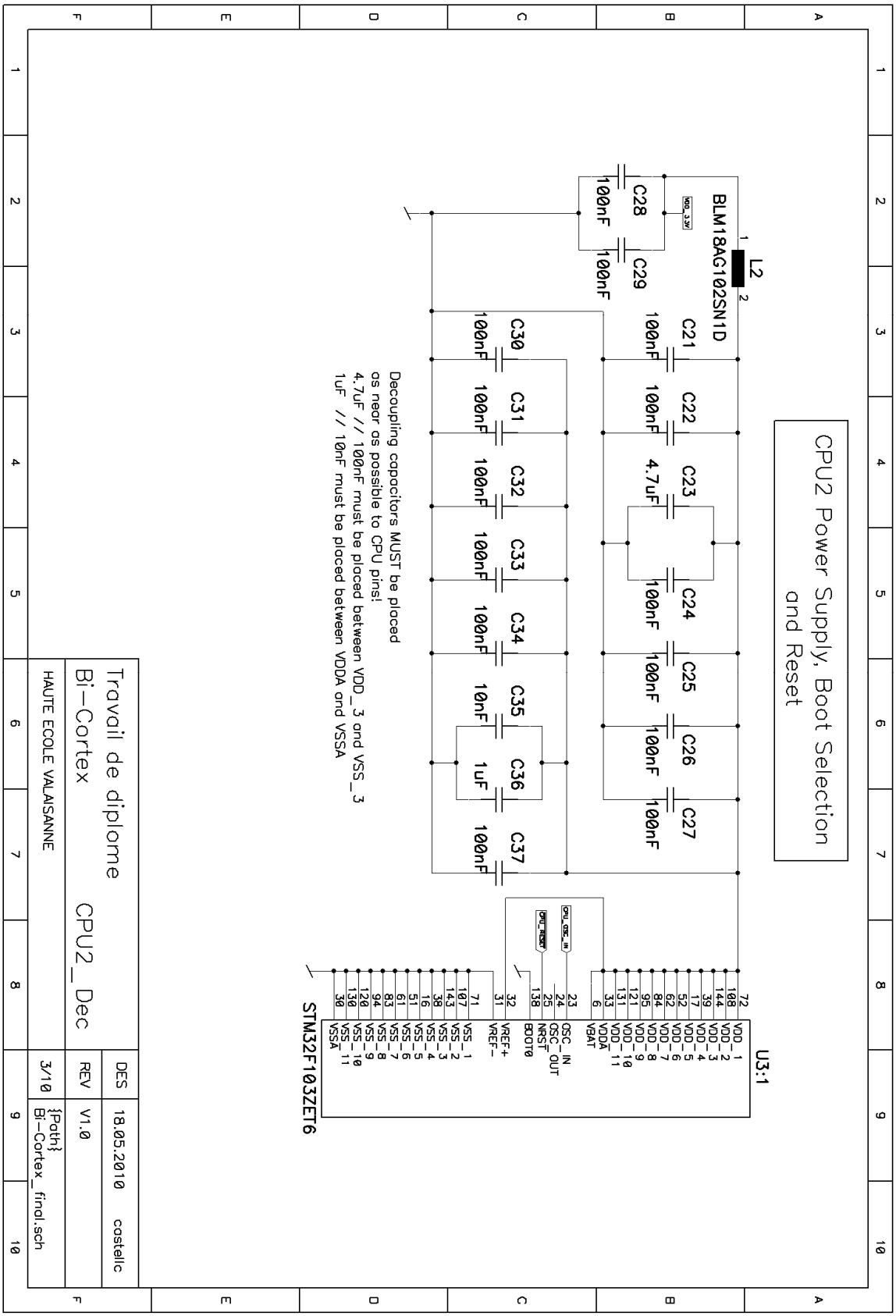
Annexe VII : schématique découplage CPU1 + Osc 8Mhz



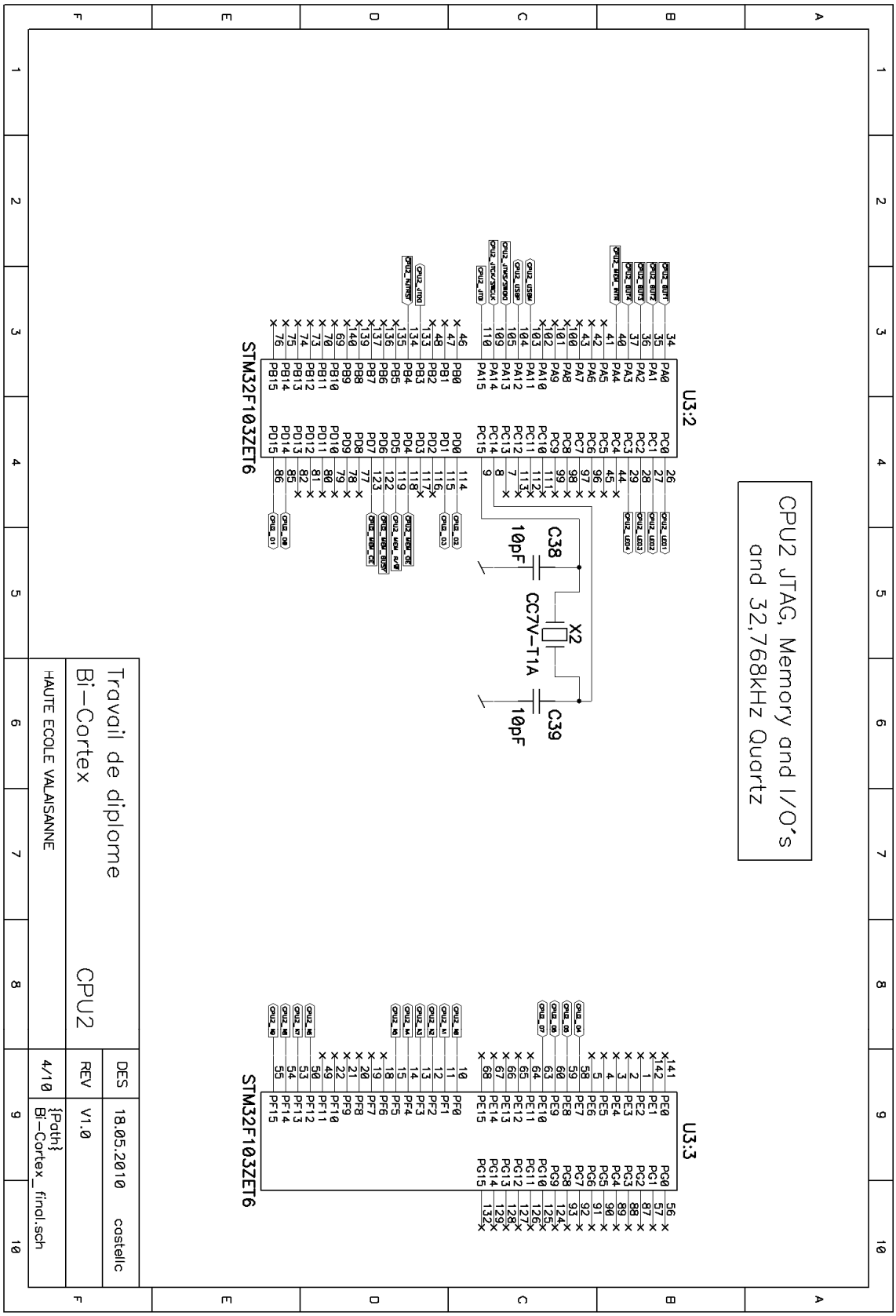
AnnexeVIII: schématique périphériques CPU1



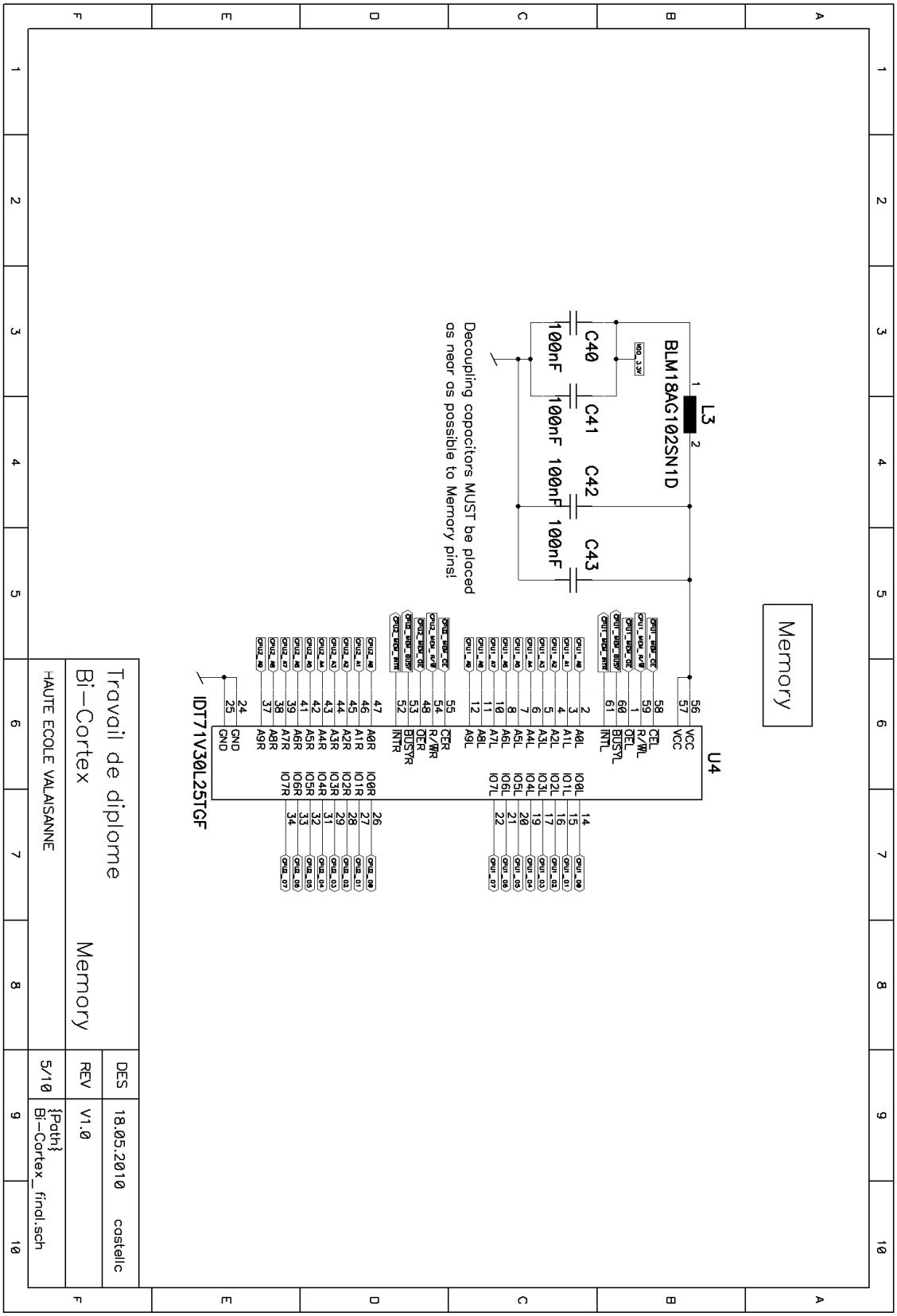
Annexe IX: schématique découplage CPU2



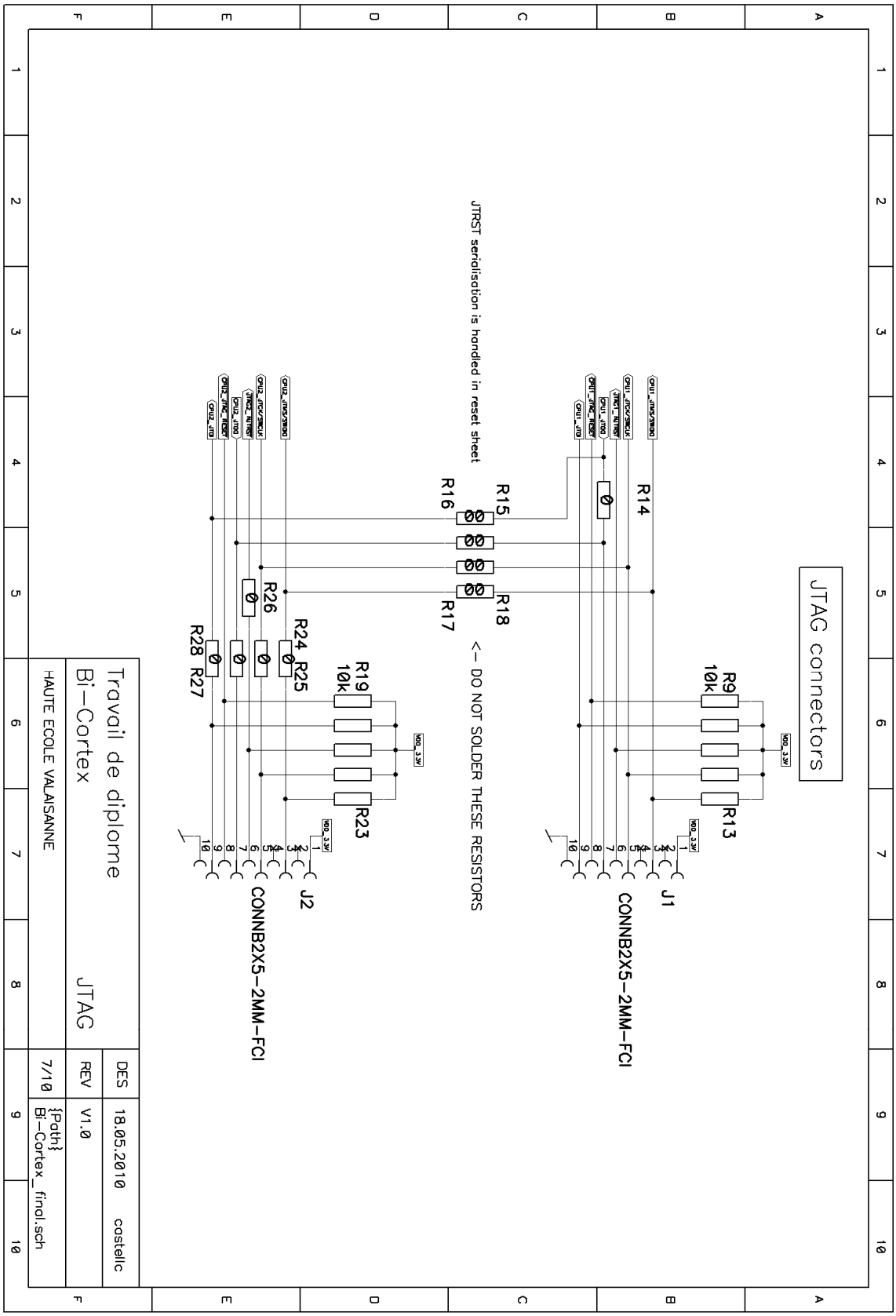
Annexe X : schématique périphériques CPU2



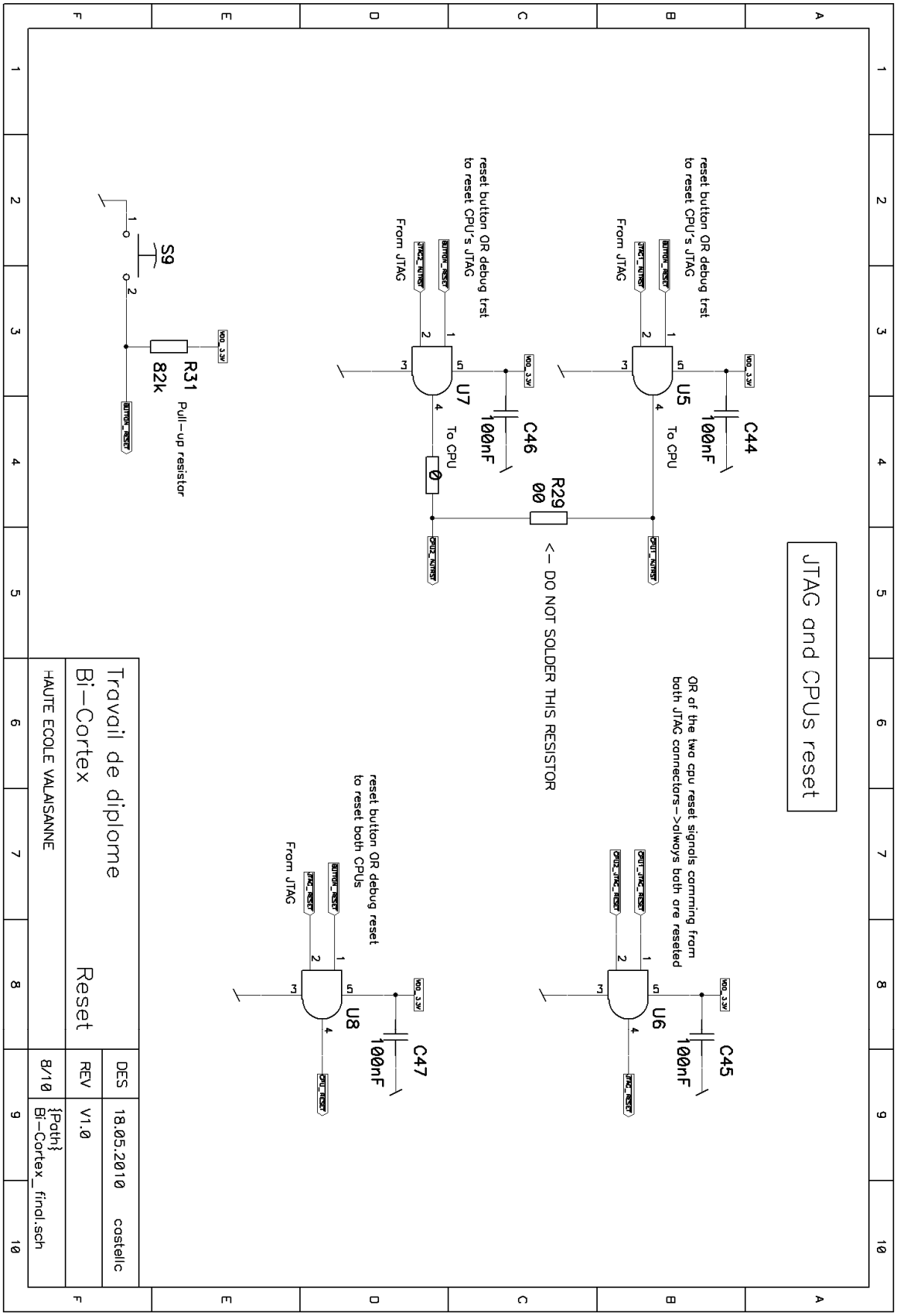
Annexe XI: schématique mémoire



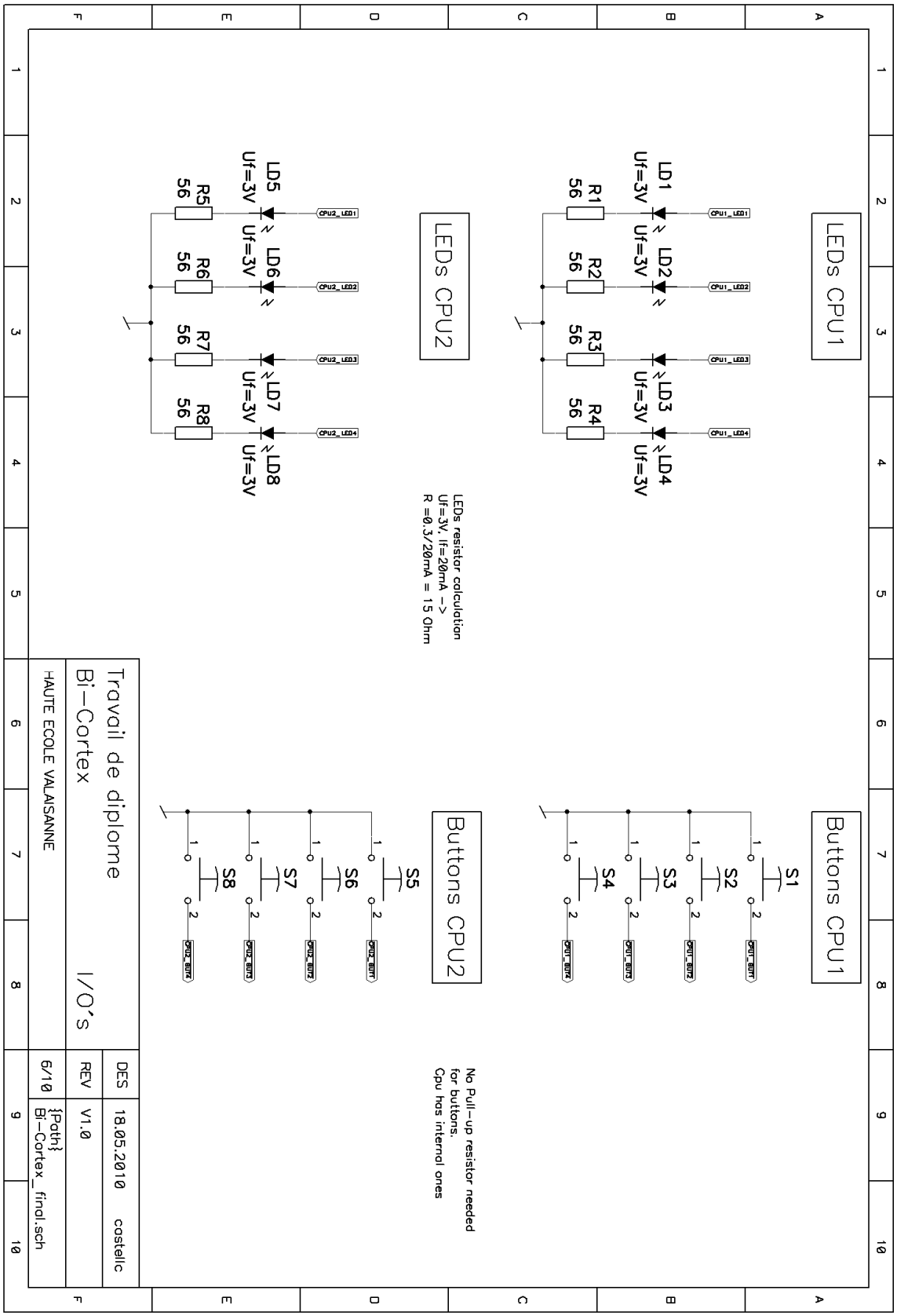
Annexe XII : schématique JTAG



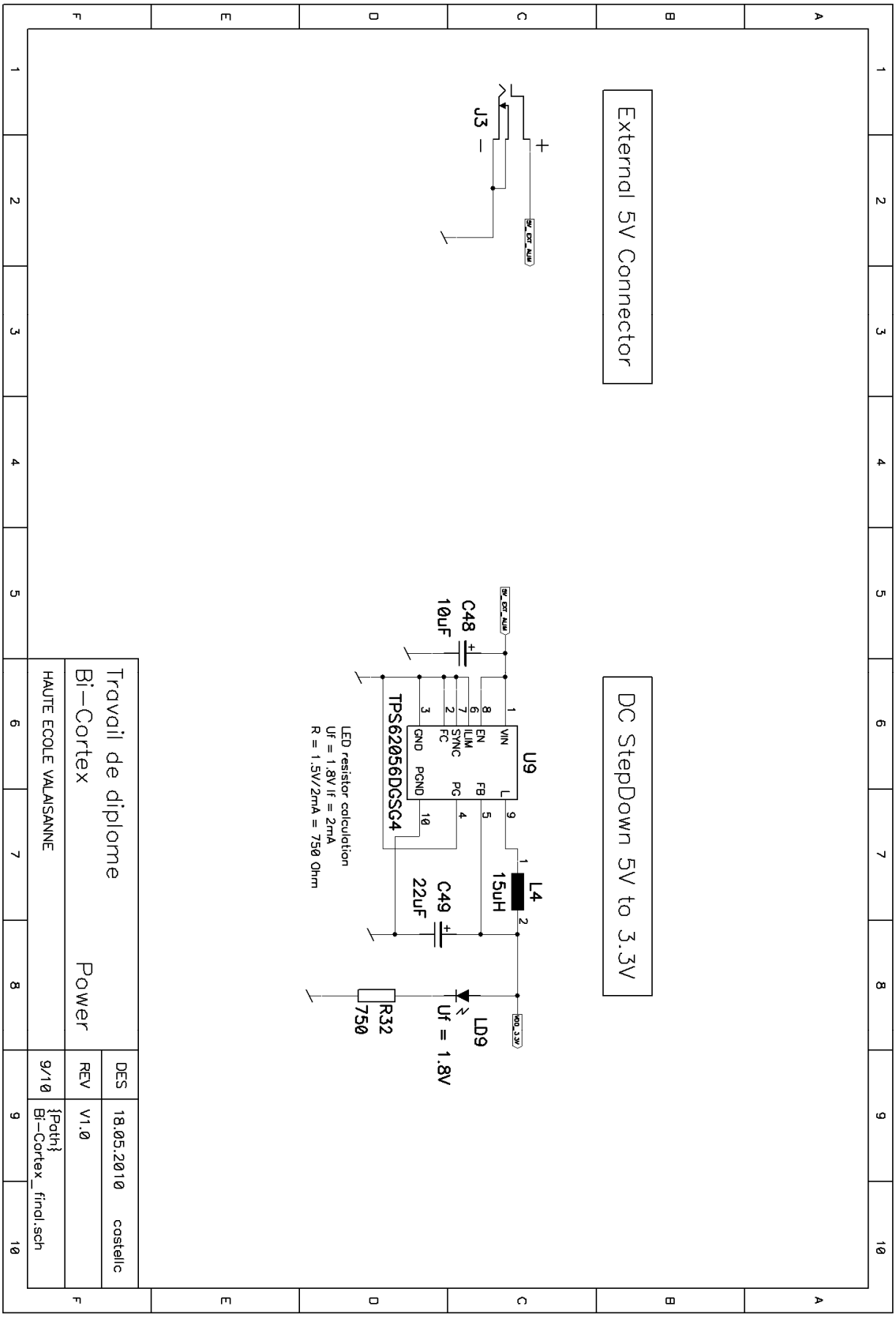
Annexe XIII : schématique reset



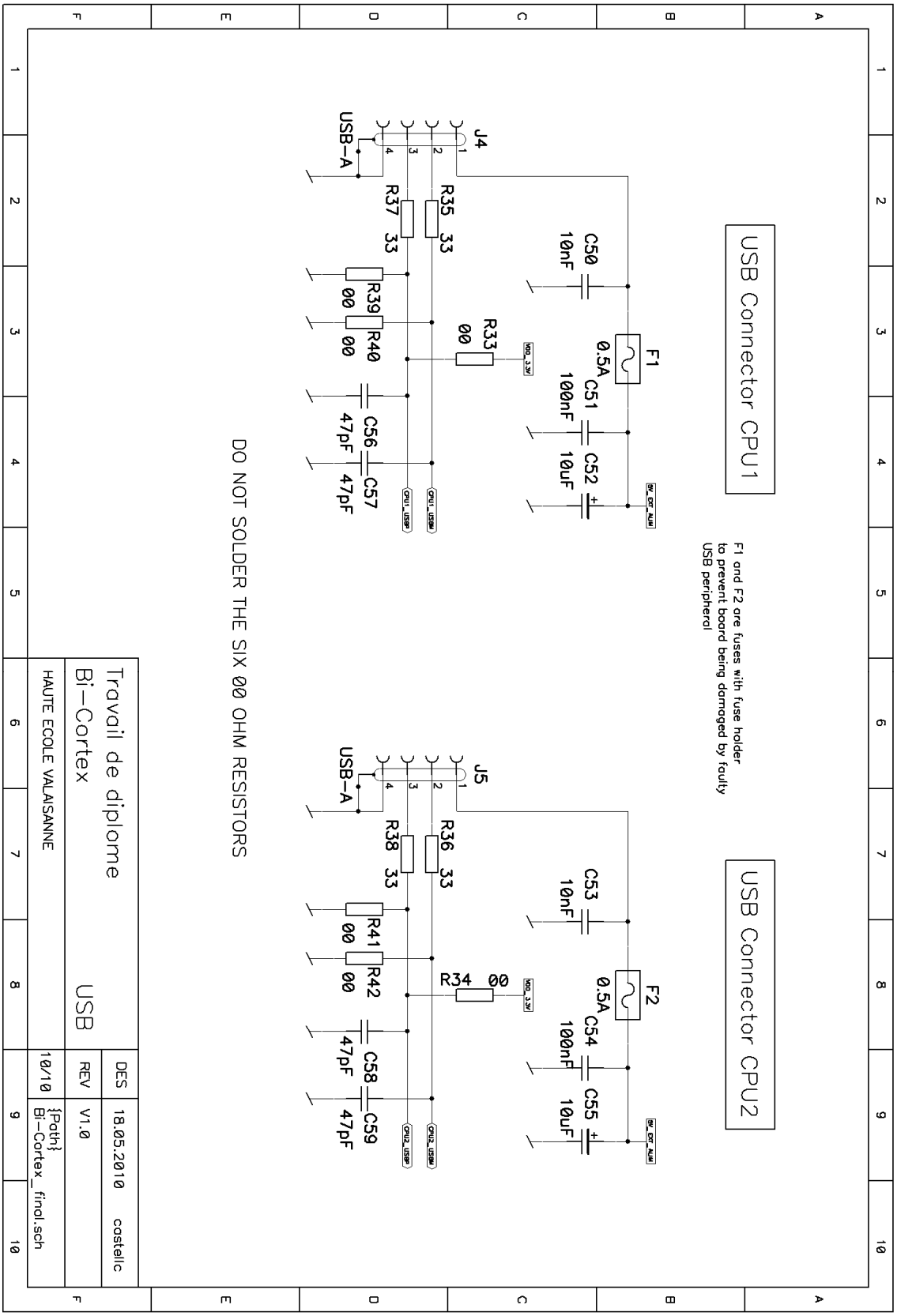
Annexe XIV : schématique I/O's



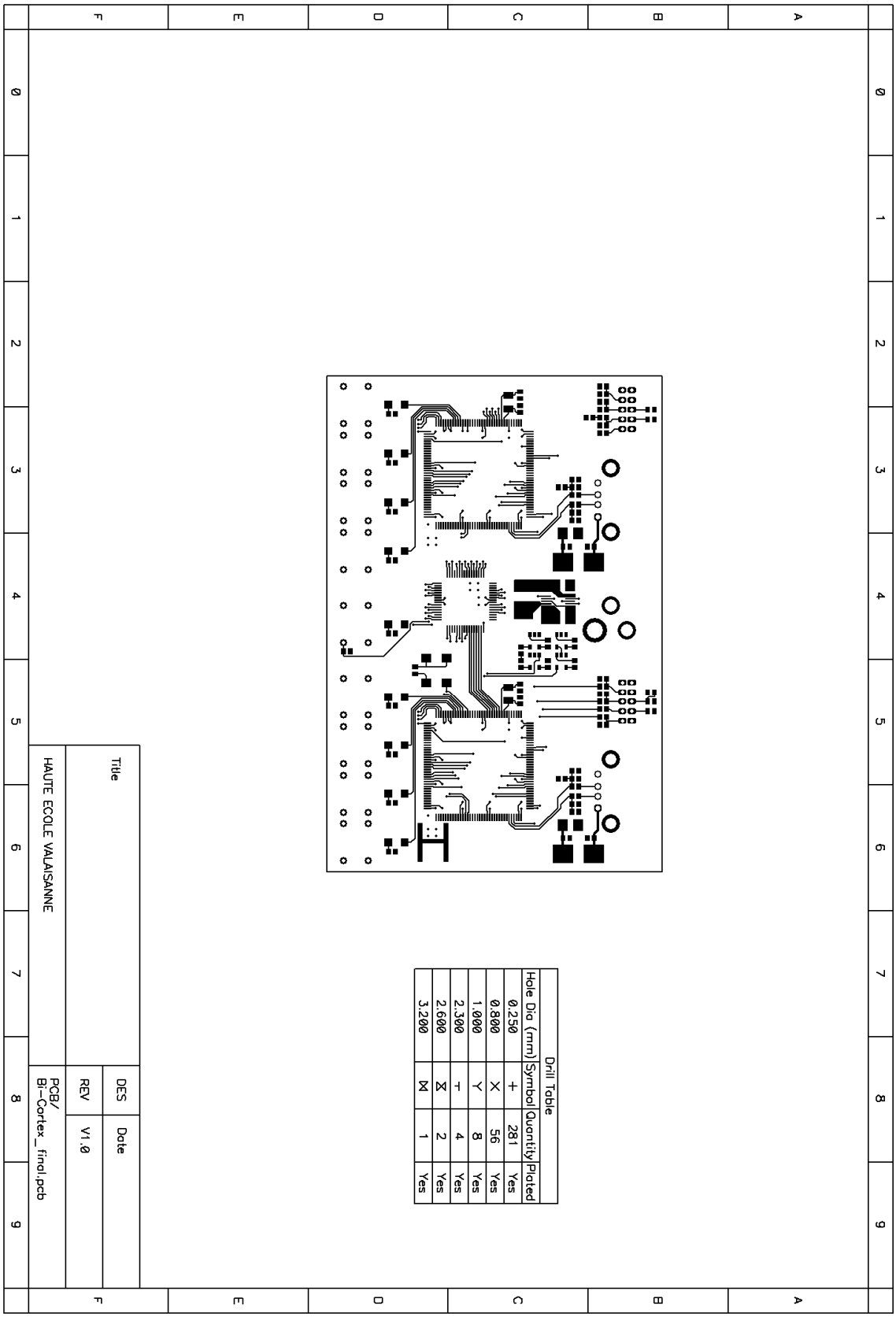
Annexe XV : schématique power



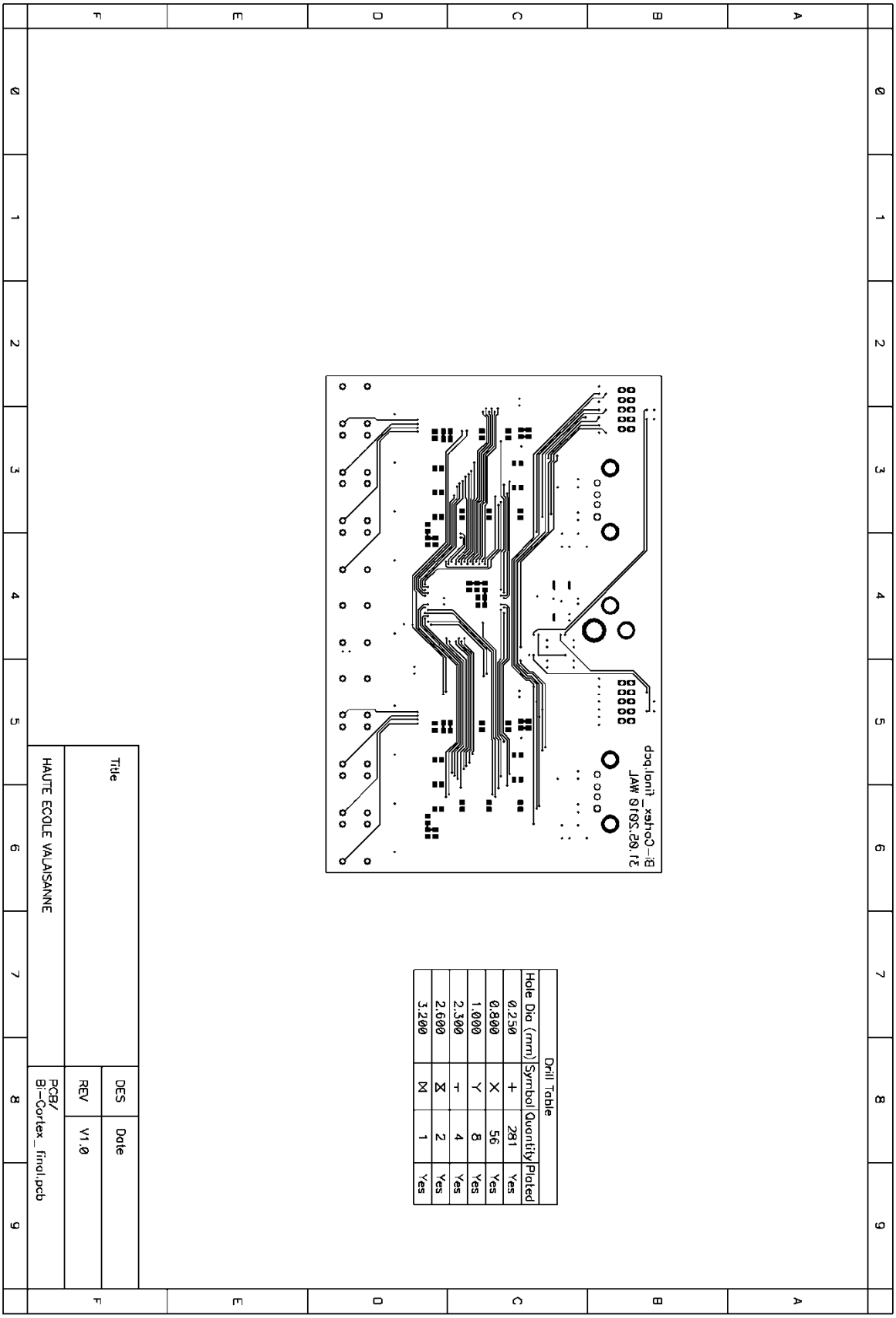
Annexe XVI : schématique USB



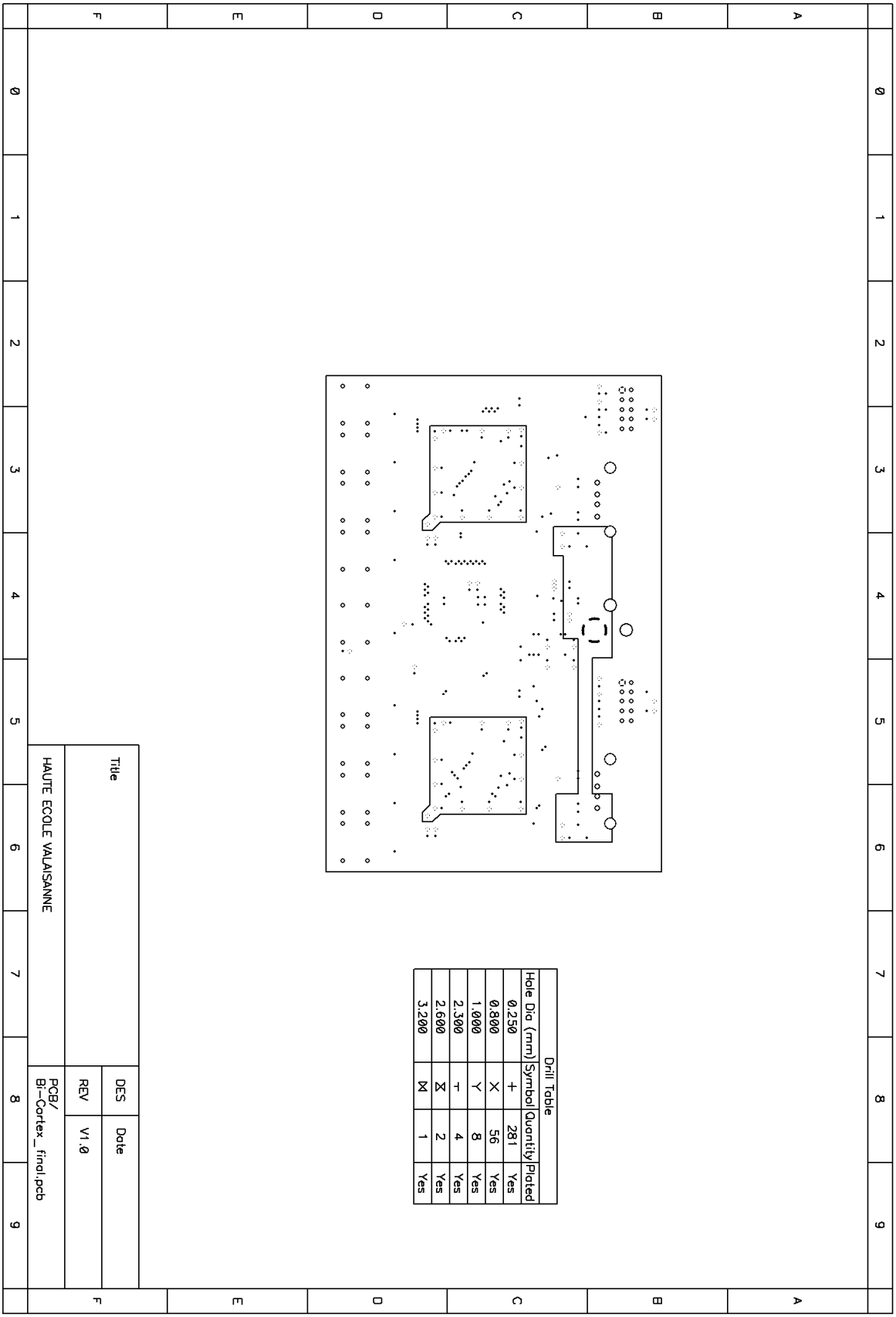
Annexe XVII : Routage PCB top



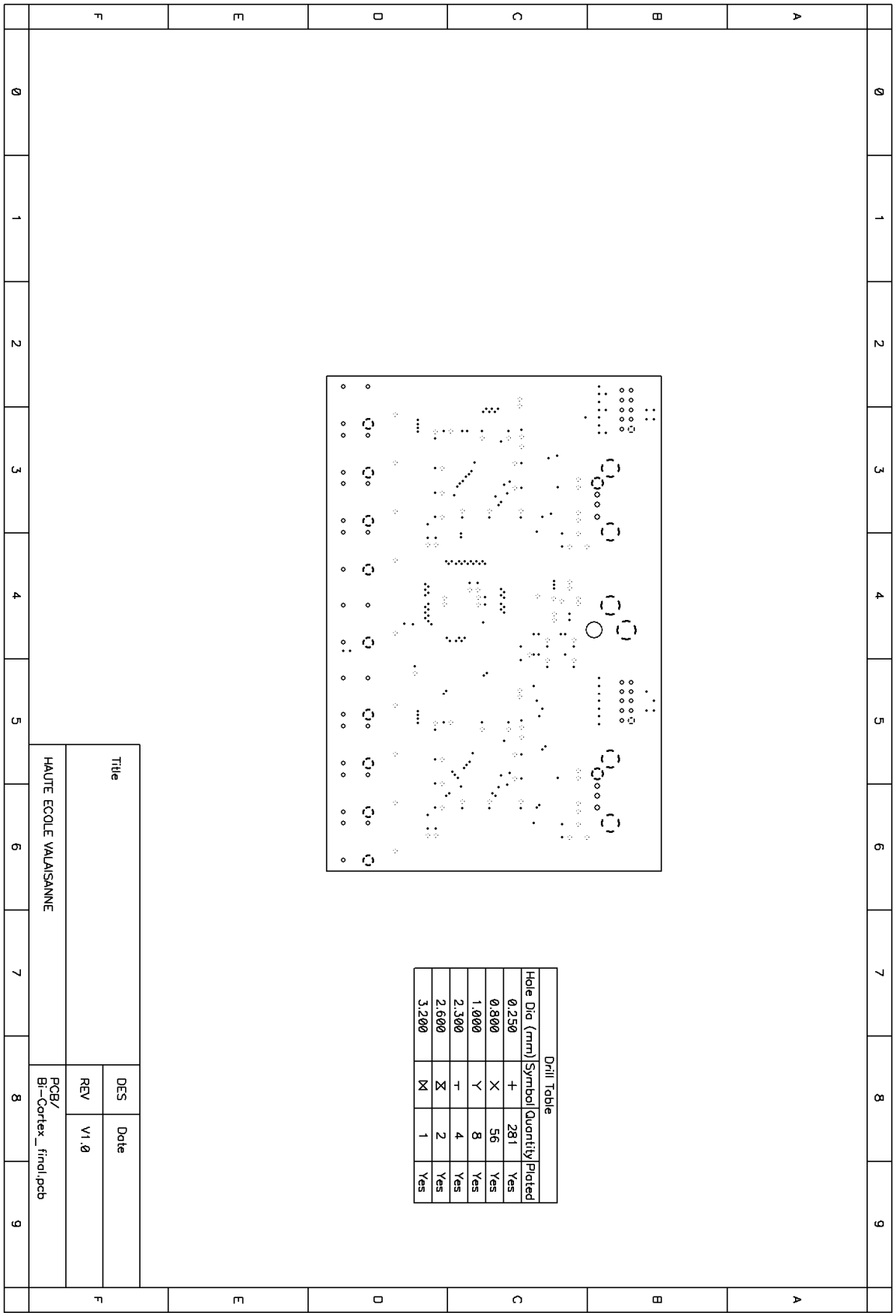
Annexe XVIII :routage PCB bottom



Annexe XIX :routage PCB VCC



Annexe XX : routage PCB gnd



Annexe XXI : liste de commande avec prix et références aux datasheets

FARNELL					
QTY	art.	Designation	price	total	Datasheet
2	1624145	STMICROELECTRONICS - STM32F103ZET6 - M	23.75	47.5	numerous in folder
1	1688284	INTEGRATED DEVICE TECHNOLOGY - IDT71V3	41.05	41.05	IDT71V30SL.pdf
5	1515671	MURATA - BLM18AG102SN1D - INDUKTIVITAET	0.114	0.57	ferrite.pdf
1	1640937	EUROQUARTZ - 8.000MHZ XO91050UITA - OSC,	8.15	8.15	osc_8MHZ.pdf
2	1539382	MICRO CRYSTAL - CC7V-T1A 32.768KHZ +-20P	3.55	7.1	quartz_32khz.pdf
4	1607780	FAIRCHILD SEMICONDUCTOR - NC7S08M5X - I	0.323	1.292	tinygateAND.pdf
3	1658503	AVX - TPSB106K016R0500 - CAPACITOR, CASE	1.8	5.4	capa10uF_lowesr.pdf
5	1135161	AVX - NOSB226M004R0600 - KONDENSATOR,	0.322	1.61	capa22uF_lowesr.pdf
5	1717485	PANASONIC - ELL6UH150M - DROSSEL, POWE	1.4	7	power_indict.pdf
1	9324305	TEXAS INSTRUMENTS - TPS62056DGSG4 - STE	8.55	8.55	stepdown70928.pdf
2	1177883	LUMBERG - 2410 02 - BUCHSE, USB, PCB TYP	1.05	2.1	usb_conn.pdf
2	1651781	LITTELFUSE - R154.750 - FUSE HOLDER, CHIP	4.65	9.3	fuse_holder.pdf
2	1703147	LITTELFUSE - R451.500L - PATRONENSICHERU	2.4	4.8	0.5_fuse.pdf
10	1572625	KEMET - C0603C475K8PAC 7867 - CAPACITOR	0.67	6.7	capa4-7uF.pdf
				total	151.122

Annexe XXII : arbre des horloges du processeur

Figure 8. Clock tree